POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria oprogramowania, procesów biznesowych i baz danych

Dominik Zdziarski
Nr albumu 28821

Uniwersalny system rekomendacji treści zrealizowany w architekturze mikroserwisów

Praca magisterska
napisana pod kierunkiem
dr inż. Mariusza Trzaski

Warszawa, lipiec 2024

Streszczenie

Praca omawia popularne metody implementacji systemów rekomendacji treści, w tym te oparte na informacjach o interakcjach użytkowników oraz podobieństwach między nimi. Przedstawione są również istniejące biblioteki i pakiety do wykorzystania z językiem programowania Python. Omówione są problemy związane z problemem zimnego startu. Praca zawiera przykładową implementację uniwersalnego systemu rekomendacji treści w architekturze mikroserwisów z opcjami zarządzania za pomocą interfejsu REST API. Na końcu przedstawione są mechanizmy weryfikacji skuteczności stworzonego oprogramowania z wykorzystaniem kroswalidacji i tablicy pomyłek.

Słowa kluczowe: systemy rekomendacji treści, personalizacja, uczenie maszynowe, wielkie zbiory danych, mikrousługa.

Spis treści

1. Wstęp

W dzisiejszych czasach, gdy dostępność treści jest nieograniczona, a ilość danych i informacji rośnie w tempie wykładniczym, utrzymanie uwagi użytkownika na odpowiednio długo staje się kluczowe dla wielu firm i organizacji. Wśród zalewu informacji, który codziennie dociera do nas z różnych źródeł, wybór tych, które są dla nas najbardziej interesujące i ważne staje się wyzwaniem. Nic więc dziwnego, że chętnie polegamy na sugestiach co przeczytać, obejrzeć lub posłuchać. Część z tych sugestii pochodzi przykładowo od znajomych, których upodobania mogą być zbliżone do naszych. Inne mogą być dziełem marketingu nastawionego na sprzedanie nam konkretnej usługi lub produktu. Jeszcze inne mogą być same w sobie usługą lub jej częścią, jak ma to miejsce w popularnych platformach streamingowych, portalach z newsami lub choćby wyszukiwarkach podróży.

1.1 Systemy rekomendacji treści

Systemy rekomendacji treści (ang. *recommendation system*, *recommendation engine*, *RecSys*) to oprogramowanie, które zwraca użytkownikowi spersonalizowane rekomendacje w oparciu o wcześniej zebrane informacje [92]. Do takich należeć może aktywność użytkownika na portalu lub jego charakterystyka. Wynikiem mogą być między innymi sugerowane produkty w sklepie internetowym, sugestie następnego filmu do obejrzenia lub polecane artykuły do przeczytania. Takie oprogramowanie jest niezwykle cenne z biznesowego punktu widzenia. Pozwala na zatrzymanie użytkownika w aplikacji lub na stronie internetowej na dłużej, co może poskutkować pobraniem opłaty za subskrypcję, produkt lub wyświetlenie kolejnej reklamy.

Problem implementacji systemów rekomendacji jest interdyscyplinarny. Ich rozwój zwykle wymaga współpracy specjalistów z różnych dziedzin: nauczania maszynowego, przetwarzania danych, inżynierii oprogramowania oraz analizy biznesowej. Metody implementacji obejmują modelowanie oparte na treści (ang. *content-based filtering*) [92], kolaboracyjne filtrowanie (ang. *collaborative filtering*) [60] oraz modele hybrydowe [24]. Na przestrzeni lat powstało wiele rozwiązań, które usprawniają implementację rekomendacji w aplikacjach. Grafowe bazy danych, takie jak Neo4j, oferują obszerną dokumentację dotyczącą budowy systemów rekomendacji treści działających w czasie rzeczywistym [75]. Biblioteki języka programowania Python: scikit-learn [96] i Surprise [101] zawierają całe klasy algorytmów służące do tworzenia spersonalizowanych rozwiązań.

1.2 Cel pracy i założenia

Celem niniejszej pracy jest analiza porównawcza metod i algorytmów rekomendacji treści oraz implementacja prototypu systemu pod postacią mikrousługi wraz z interfejsem REST API. Analiza obejmuje następujące klasy algorytmów:

1. Modele oparte na treści (ang. *content-based filtering*).
2. Modele kolaboracyjne (ang. *collaborative filtering*).
3. Modele wykorzystujące sieci neuronowe głębokiego uczenia (ang. *deep neural network*) [81].
4. Modele hybrydowe.

Zakres pracy obejmuje także:

1. Przegląd literatury dotyczącej algorytmów rekomendacji spersonalizowanych treści.

2. Implementację i porównanie skuteczności wybranych algorytmów, z wykorzystaniem kroswalidacji.

3. Przygotowanie intuicyjnego interfejsu graficznego użytkownika, który umożliwia interakcję z mikrousługą przez użytkowników nietechnicznych.

4. Publikację obrazu mikrousługi na Docker Hub, z instrukcją uruchomienia.

1.3 Wykorzystane technologie

Projekt został przygotowany w oparciu o następujące technologie:

1. Analizowane algorytmy zostały uruchomione w Pythonie 3.10.6 [87].

2. Kroswalidacja przeprowadzona została z użyciem pakietu scikit-learn. [96].

3. Mikrousługa została napisana w języku programowania Python z wykorzystaniem technologii Flask [79] w wersji 2.3.2 według zasad REST API. Komunikacja z bazą danych odbywa się za pomocą biblioteki Psycopg [27] w wersji 2.9.

4. Wykorzystane bazy danych to PostgreSQL w wersji 15.2. [102].

5. Repozytorium z kodem źródłowym jest przechowywane w systemie kontroli wersji Git na platformie GitHub. [41].

6. Komunikacja z mikrousługą odbywa się przy użyciu klienta HTTP Insomnia. [57].

1.4 Organizacja pracy

Pierwsza część pracy stanowi przegląd aktualnego stanu wiedzy w zakresie metod implementacji systemów rekomendacji. W przeglądzie uwzględniono zarówno podejścia bazujące na wcześniej zgromadzonych danych o interakcjach użytkowników, jak i metody oparte na podobieństwie cech charakterystycznych. W tej sekcji zaprezentowano również przydatne biblioteki i pakiety, ze szczególnym uwzględnieniem tych kompatybilnych z językiem Python 3. Ponadto, omówiono wyzwania związane z implementacją systemów rekomendacji, takie jak problem zimnego startu (ang. *cold start problem*) [61], polegający na niewystarczającej ilości danych do wygenerowania spersonalizowanych rekomendacji dla nowych użytkowników.

Kolejna część zawiera przykładową implementację uniwersalnego systemu rekomendacji treści, zrealizowaną w architekturze mikroserwisów. Zaprojektowana mikrousługa udostępnia interfejs REST API, umożliwiający efektywne zarządzanie systemem. W pracy zaprezentowano schemat bazy danych, warstwę dostępu do danych oraz kontrolery, a także omówiono architekturę systemu. Zaimplementowana mikrousługa została udostępniona w postaci obrazu na platformie Docker Hub, co ułatwia jej szybkie wdrożenie i uruchomienie.

Ostatnia część pracy koncentruje się na metodach oceny efektywności stworzonego oprogramowania, wykorzystujących techniki kroswalidacji oraz klasyfikacji z użyciem tablicy pomyłek (ang. *confusion matrix*). Kroswalidacja polega na podzieleniu analizowanego zbioru danych na podzbiory [40], a następnie przeprowadzeniu testów na wydzielonym podzbiorze testowym, podczas gdy pozostałe dane służą jako zbiór treningowy. Takie podejście umożliwia weryfikację skuteczności systemu rekomendacji dla różnych zestawów danych testowych, minimalizując jednocześnie ryzyko zniekształcenia wyników przez nadmierne dopasowanie do danych treningowych.

2. Biznesowe potrzeby i motywacja

Historycznie systemy rekomendacji treści powstały jako odpowiedź na potrzeby biznesowe w erze dotcomów, zwaną także bańką dotcom (ang. *dot-com bubble*), na przełomie lat 90. i 2000 [69]. Wykorzystywane w celu ułatwienia dostępu do treści oraz zwiększenia dochodów właścicieli tych treści, systemy te mogą znacznie zwiększyć ilość użytkowników, którzy opłacają dostęp do zagregowanych danych lub kupują polecane produkty. Systemy rekomendacji treści są wykorzystywane także, by lepiej zaplanować dalszy rozwój firmy przez zbieranie stosownych metryk, na podstawie których podjęte zostaną kolejne decyzje biznesowe.

2.1 Efektywne zarządzanie dokumentami

Badania nad systemami rekomendacji są stosunkowo nowe w porównaniu z badaniami nad innymi technologiami systemów informatycznych [92]. Ich historia sięga lat 90. XX wieku. W ciągu kolejnych lat powstały pierwsze systemy rekomendacji, które miały na celu polecanie użytkownikom produktów muzycznych i filmów. W 1992 roku, w Xerox PARC, stworzono system rekomendacji maili w grupach dyskusyjnych o nazwie Tapestry [32]. Wraz z tym systemem do szerszej świadomości wdarł się termin filtrowania kolaboratywnego (ang. *collaborative filtering*). W tym samym roku, w Stanach Zjednoczonych, Paul Resnick i John Riedl założyli na Uniwersytecie Minnesoty firmę GroupLens Research, zajmującą się profesjonalnymi badaniami w dziedzinie systemów rekomendacji treści [58]. Wkrótce potem, w 1994 roku, firma opublikowała wyniki prac nad systemem filtrowania newsów opartym na sieci Usenet, oferującym w porównaniu do Tapestry dodatkowe możliwości dostosowane do rozproszonych sieci oraz zapewniającym skalowalność. Nowe rozwiązanie, podobnie jak poprzednik, generowało sugerowane treści w oparciu o oceny użytkowników.

Kolejne lata przyniosły zainteresowanie ze strony biznesu, który dostrzegł potencjał sprzedażowy w spersonalizowanych treściach. Pierwszą firmą, która komercyjnie udostępniała usługi personalizacji, była *Net Perceptions*, założona w 1996 roku przez pracowników GroupLens w trakcie okresu zwanego bańką dotcom (ang. *dot-com bubble*) [22]. Firma odpowiedziała na bieżące potrzeby i dostarczała podmiotom komercyjnym pokroju Amazon swoje rozwiązania [15].

Pomimo przejęcia i zniknięcia firmy Net Perceptions z rynku, jej pierwotny twórca - grupa badawcza GroupLens - kontynuuje swoją działalność. Na stronie internetowej tej grupy można znaleźć informacje o aktualnie prowadzonych projektach, w tym o MovieLens [46] - internetowym systemie rekomendacji, którego zbiory danych są regularnie publikowane bezpłatnie w internecie.

Istnieją konferencje i warsztaty poświęcone wyłącznie tej dziedzinie, takie jak seria konferencji Association for Computing Machinery poświęconych systemom rekomendacji. Konferencja ta jest głównym corocznym wydarzeniem poświęconym badaniom i zastosowaniom RecSys [92]. Ponadto sesje poświęcone systemom rekomendacji są często uwzględniane w bardziej tradycyjnych konferencjach z zakresu baz danych, systemów informatycznych, modelowania użytkowników, sztucznej inteligencji, uczenia maszynowego, nauki o danych i systemów adaptacyjnych.

2.2 Zwiększenie ilości wypożyczanych filmów

Znaczenie problemu rekomendacji spersonalizowanych treści podkreśla również fakt, że w 2006 roku firma Netflix ogłosiła konkurs na usprawnienie swojego algorytmu rekomendacji filmów [17]. Warto wspomnieć, że w tamtym czasie firma zajmowała się jeszcze wypożyczaniem płyt z filmami. Uczestnicy zorganizowanego konkursu zestawiali swoje implementacje względem autorskiego algorytmu Netflixa o nazwie Cinematch. Główną nagrodą był milion dolarów. W konkursie wzięli udział naukowcy, profesjonalni analitycy i hobbystyczni programiści, którzy chcieli sprawdzić swoje umiejętności [64]. Zestaw

danych (ang. *data set*) pobrało ponad 30 tysięcy entuzjastów, a zwycięski zespół *BellKor's Pragmatic Chaos*, złożony z pracowników wielu firm, w tym AT&T oraz Yahoo!, uzyskał wynik wyższy od Cinematch o 10.06% [76].

Choć w kolejnych latach zaprzestano organizowania konkursu z uwagi na problemy z prywatnością danych [99], firma Netflix w dalszym ciągu udoskonala systemy rekomendacji wewnątrz firmy, tym razem na potrzeby rekomendacji odbywających się online w usłudze streamingowej. Dział firmy Netflix Research udostępnił publicznie dokumenty [89], z których można wywnioskować, że na istotną część skomplikowanej technologii i modeli uczenia maszynowego składają się między innymi algorytmy:

1. Uczenia posiłkowego przez wzmacnianie (ang. *reinforcement learning*) - uczenie ze wzmocnieniem to rodzaj algorytmu uczenia maszynowego, w którym agent uczy się podejmować decyzje metodą prób i błędów w środowisku, które nagradza lub karze te decyzje [54]. Model działa poprzez interakcję ze środowiskiem, gromadząc doświadczenia, na podstawie których poprawia swoją wydajność w celu maksymalizacji sumy nagród.

2. Sieci neuronowej (ang. *neural network*) - sieci neuronowe są inspirowane strukturą neuronalną ludzkiego mózgu i składają się z wielu połączonych ze sobą węzłów, które przetwarzają dane wejściowe i generują odpowiednie dane wyjściowe [81]. Sieci te są szczególnie przydatne do analizy wzorców i trendów, które są zbyt złożone dla tradycyjnych algorytmów.

3. Modeli przyczynowo-skutkowych (ang. *causal model*) - modele przyczynowe wykorzystują metody statystyczne do analizowania i wykrywania związków przyczynowych między zmiennymi. Modele te pozwalają wnioskować o tym, jakie skutki mogą wynikać z określonych przyczyn [53].

4. Faktoryzacji macierzy (ang. *matrix factorization*) - to proces rozkładu jednej dużej macierzy na iloczyn dwóch lub więcej mniejszych macierzy [73]. Jest to powszechnie stosowana metoda w systemach rekomendacji, gdzie głównym celem jest identyfikacja ukrytych czynników charakteryzujących zarówno użytkowników, jak i produkty.

2.3 Dopasowanie utworów do gustu słuchaczy

Inne przedsiębiorstwa także z powodzeniem wdrożyły systemy rekomendacji treści. Spotify to platforma muzyczna z ponad 75 milionami aktywnych użytkowników [35]. Oferuje ona wiele mechanizmów personalizacji. Co tydzień utwory na playliście Discover Weekly są zmieniane, a codziennie użytkownicy otrzymują kilka propozycji playlist w ramach Daily Mix. Specjalna lista o nazwie Release Radar zawiera najnowsze utwory od artystów, których śledzi użytkownik, a także nowości, wybrane specjalnie dla niego. Lista jest aktualizowana co piątek. Ponadto, utwory, które są dobierane także po zakończeniu słuchania wybranych playlist. W ten sposób zamiast kończyć odtwarzanie kolejkowane są następne utwory.

W 2023 roku Dietmar Jannach i Himan Abdollahpouri z działu R&D Spotify stwierdzili, że dokładność przewidywań rekomendacji nie zawsze jest wystarczająca dla uznania systemu za użyteczny [51]. Przykładowo, sugerowanie kolejnej części sagi *Gwiezdne wojny* zagorzałemu fanowi może być trafne, z marginesem błędu bliskim zeru, ale istnieje prawdopodobieństwo, że użytkownik już zna ten film. Autorzy określają skuteczność współczesnych systemów rekomendacji treści jako wykraczającą poza samą dokładność (ang. *beyond-accuracy*). Oprócz precyzji, Dietmar i Himan definiują szereg innych kluczowych metryk i celów, które powinny przyświecać twórcom systemów rekomendacji treści:

1. Cele jakościowe (ang. *quality objectives*) - skuteczność, różnorodność i zdolność do sugerowania nowych treści.

2. Cele interesariuszy - sugestie, które nie zawsze są najlepsze dla użytkownika, mogą być bardziej opłacalne dla przedsiębiorstwa.

3. Długoterminowe cele - rekomendacje mogą wpływać na zaangażowanie i trendy zarówno krótko-, jak i długoterminowo.

4. Cele związane z doświadczeniem użytkownika (ang. *user experience, UX*) - ilość i jakość rekomendacji powinna wpasować się w całość doświadczeń UX.

5. Cele i ograniczenia inżynieryjne - nowoczesne systemy uczenia maszynowego mogą być kosztowne w rozwoju i utrzymaniu.

2.4 Zachęcenie czytelników do kolejnego artykułu

Wydawca prasy i mediów, firma Ringier Axel Springer Polska, posiadająca w swoim portfolio portale informacyjne i publicystyczne, między innymi takie jak Onet lub Forbes Polska, dociera codziennie do milionów użytkowników [93]. W interesie przedsiębiorstwa jest zachęcenie czytelników do sprawdzenia *jeszcze jednego artykułu*. W 2020 roku, pracownicy firmy na blogu opublikowali artykuł, dzieląc się powodem, dla którego wybrano rekomendacje w oparciu o segmentację użytkowników [74]. Klasyczne podejście oparte na faktoryzacji macierzy jest zbyt czasochłonne dla skali operacji firmy. Zastosowane zamiast tego algorytmy wielorękiego bandyty (ang. *multi-armed bandit*) i segmentacji pozwoliły na:

1. Przeciwdziałanie rozproszeniu danych i efektywne dopasowanie treści.

2. Uniknięcie problemu bańki informacyjnej (ang. *filter bubble, information bubble*) - czyli rekomendowania wyłącznie wąskiego przekroju informacji. Zamiast tego, umożliwiło odkrywanie nowych treści.

3. Obniżenie kosztów - indywidualne rekomendacje są droższe z perspektywy mocy obliczeniowej niż te skierowane do segmentów użytkowników.

4. Lepsze zrozumienie danych analitycznych jest możliwe dzięki wyodrębnieniu tematów z segmentów. Pozwala to nie tylko zrozumieć zainteresowania czytelników, ale także charakterystykę popularnych treści.

2.5 Motywacja

Wspomniane przypadki to zaledwie kropla w morzu opisująca wykorzystanie systemów rekomendacji treści. Motywacją do podjęcia tematu niniejszej pracy była rosnąca popularność tego typu rozwiązań, która, w połączeniu z coraz większą mocą obliczeniową chmur obliczeniowych, pozwala na szybsze, tańsze i dokładniejsze działanie.

Ponadto rozwój umiejętności analitycznych, uczenia maszynowego i sztucznej inteligencji, w połączeniu ze studiowaniem metodologii badań i analizą literatury dziedzinowej, pozwala lepiej zrozumieć złożoność i wyzwania związane z systemami rekomendującymi. Znajomość tego tematu umożliwia praktyczne zastosowanie wiedzy w różnych sektorach rynku, od e-commerce po usługi streamingowe, oferując rozwiązania, które zmieniają sposób odkrywania i konsumowania treści.

Praca ta pozwala także zbadać etyczne aspekty systemów rekomendacji, które zyskują na znaczeniu w cyfrowym świecie, gdzie algorytmy wpływają na opinie i zachowania ludzi. Możliwość przyczynienia się do rozwijania bezpieczniejszych, bardziej transparentnych i odpowiedzialnych technologii rekomendacyjnych jest kolejnym ważnym aspektem tej pracy.

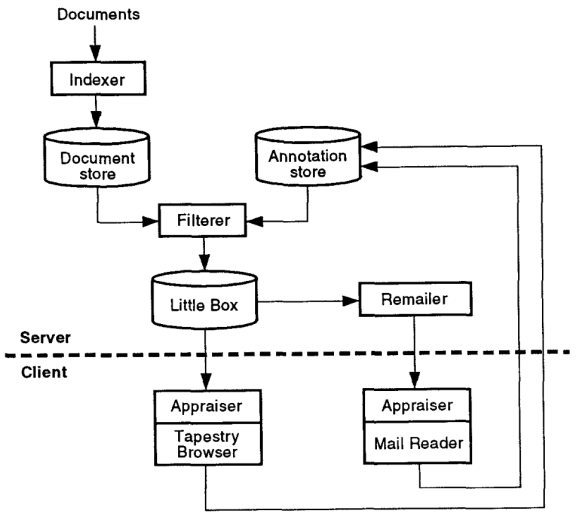
3. Stan sztuki

Niniejszy rozdział szczegółowo opisuje rozwój systemów rekomendacji. Rozpoczynając od projektu Tapestry w firmie Xerox PARC, który zapoczątkował wykorzystanie filtrowania kolaboracyjnego, przechodzimy do nowoczesnych implementacji. Zostaną omówione inicjatywy takie jak konkurs Netflix Prize, który spopularyzował globalne zainteresowanie doskonaleniem algorytmów rekomendacji filmów. Szczególną uwagę poświęcono ewolucji algorytmów stosowanych przez Amazon, które ewoluowały od prostych systemów do zaawansowanych rozwiązań hybrydowych. Dalsza część rozdziału koncentruje się na systemach rekomendacji w serwisach takich jak Spotify, GoodReads, Reddit, YouTube i TikTok, które dostosowują treści do preferencji użytkowników.

3.1 Tapestry

Rozwój systemów rekomendacji treści zyskał na znaczeniu na początku lat 90. XX wieku kiedy to firma Xerox PARC (Xerox Palo Alto Research Center) zainicjowała projekt Tapestry [42]. Był to systemem rekomendacji maili w grupach dyskusyjnych, który wprowadził koncepcję filtrowania grupowego, stanowiącą kamień milowy dla przyszłych systemów rekomendacji. Ten przełomowy projekt położył podwaliny pod rozwój technologii, która obecnie odgrywa kluczową rolę w odkrywaniu i korzystaniu z treści cyfrowych, od e-commerce po platformy streamingowe.

Tapestry powstało w odpowiedzi na rosnące wykorzystanie poczty elektronicznej, które przytłaczało użytkowników ogromną ilością przychodzących dokumentów [43]. Rozwiązania takie jak listy mailingowe, choć użyteczne, często nie były wystarczająco elastyczne, aby zaspokoić indywidualne potrzeby użytkowników w zakresie filtrowania treści. W związku z tym w Xerox PARC opracowano eksperymentalny system Tapestry wspierający wymianę poczty elektronicznej, który miał na celu zaoferowanie bardziej skutecznego sposobu na filtrowanie i zarządzanie dużymi ilościami maili. Kluczowym elementem Tapestry było zaangażowanie ludzi w proces filtrowania przez zapisywanie ich reakcji na dokumenty, które czytali. Te reakcje, znane jako adnotacje (ang. *annotations*), mogły być wykorzystywane przez filtry innych użytkowników, co prowadziło do efektywniejszego i spersonalizowanego filtrowania treści opartego na współpracy użytkowników, znanego jako filtrowanie kolaboracyjne inaczej zwane grupowym [91].



Rysunek 3.1: Architektura systemu Tapestry. Źródło: [42]

Innowacyjność Tapestry, którego architekturę przedstawiono na rysunku 3.1 polegała na wykorzystaniu relacji pomiędzy różnymi dokumentami i naniesionymi adnotacjami. Architektura programu Tapestry obejmowała moduł indeksera, który przyjmował dokument, parsował go na zestaw powiązanych informacji, wstępnie przetwarzał dane, aby następnie zapisywał dane do magazynu dokumentów (ang. *document store*). Podczas indeksacji, poszczególne wyrazy z dokumentu były zapisywane w polu *words*, czyli słowa. Pomijano wówczas powszechnie używane angielskie słowa, a formy gramatyczne wyrazów zamieniano na ich odpowiedniki podstawowe. Wykorzystano w tym celu algorytm zwany stemizacją (ang. *stemming*) [6]. Przykładowo, słowo *ran*, będące przeszłą formą czasownika biegać, zamieniano na formę podstawową *run*.

Magazyn dokumentów przechowywał dokumenty w postaci niezmiennej (ang. *immutable*) wyłącznie do odczytu. W związku z tym, adnotacje do dokumentów niosące dodatkowe informacje o przeznaczeniu lub kontekście, były zapisywane w osobnym magazynie danych na adnotacje (ang. *document annotation store*). Dane adnotacji były odseparowane od dokumentów, ponieważ mogły mieć różnorodny format i złożoną strukturę, różniącą się od struktury dokumentów [42]

Moduł nazwany rzeczoznawcą (ang. *appraiser*) służył do dopasowania dokumentu zarówno do wybranych kryteriów wyszukiwania, jak i preferencji użytkownika. Rzeczoznawca analizował i klasyfikował treści według kryteriów ustalonych przez użytkownika, takich jak tematyka, autorstwo, czy liczba otrzymanych rekomendacji. Na przykład, użytkownik mógł ustawić, że preferuje dokumenty związane z określonymi tematami lub te, które zostały wysoko ocenione przez innych zaufanych użytkowników. Rzeczoznawca, wykorzystując te preferencje, automatycznie organizował przychodzące dokumenty, nadając im odpowiednią kategorię i priorytet w skrzynce odbiorczej użytkownika. Dzięki temu procesowi, użytkownik mógł szybko i efektywnie otrzymywać informacje odpowiednie dla jego zainteresowań i potrzeb.

Małe skrzynki (ang. *little boxes*) to element architektury systemu Tapestry, który przechowywał i zarządzał dokumentami, które zostały uznane za interesujące dla poszczególnych użytkowników. Działały one jak indywidualne repozytoria, które przechowywały dokumenty, które przeszły przez pierwszy poziom filtrowania binarnego systemu. Te dokumenty, które spełniły kryteria wyszukiwania, były umieszczane w małej skrzynce. Następnie rzeczoznawca, w drugim kroku, aplikował bardziej szczegółową klasyfikację, kategoryzację i priorytetyzację. W ten sposób, małe skrzynki ułatwiały zarządzanie strumieniem informacji i dostępem do treści w systemie Tapestry.

Przeglądanie dokumentów, zwłaszcza maili, w systemie Tapestry wykorzystywało dedykowany język zapytań (ang. *Tapestry Query Language (TQL)*). Podczas projektowania systemu rozważano użycie SQL, ale ostatecznie zrezygnowano z tego ze względu na różnice między relacyjnym modelem baz danych i modelem dokumentów używanym w Tapestry. Relacyjne bazy danych posiadają ustalony zestaw kolumn, na których można wykonywać operacje, podczas gdy Tapestry posiadało dynamiczny zbiór pól opisujących dokument. Przykładowe zapytanie przedstawiono we fragmencie kodu 3.1:

Kod 3.1: Przykładowy kod w TQL

```
m.sender = 'Jan Kowalski'
AND m.subject LIKE '%PJATK%'
```

W przykładzie 3.1 pokazano jak można było pobrać wiadomości wysłane przez Jana Kowalskiego zawierające w tytule frazę *PJATK*. Choć TQL był prostym językiem w porównaniu do SQL, zespół odpowiedzialny za Tapestry przygotował dla osób nietechnicznych przeglądarkę małych skrzynek. Za pomocą interfejsu graficznego można było skorzystać z predefiniowanych zapytań.

3.2 Netflix Prize

Firma Netflix założona w 1997 roku przez Reeda Hastingsa i Marca Randolpha w Kalifornii, jest obecnie jedną z największych firm technologicznych na świecie. Zatrudnia ponad 12 800 pracowników na całym świecie i specjalizuje się obecnie w dystrybucji treści video za pomocą serwisu online. Początkowo firma zajmowała się wypożyczaniem filmów na płytach i kasetach przez stacjonarne punkty. Jeszcze zanim firma rozpoczęła transformację cyfrową, zasłynęła z konkursu dla analityków i naukowców pod tytułem Netflix Prize [17].

Reed Hastings, jeden z założycieli firmy, w duchu poprawy jakości świadczonych usług drogą otwartych konkursów społeczności, utworzył konkurs pod tytułem Netflix Prize [95]. Był to konkurs ogłoszony w 2006 roku, mający na celu poprawienie algorytmów rekomendacji filmów i programów w ich usłudze streamingowej. To wydarzenie przyciągnęło uwagę naukowców, inżynierów oraz entuzjastów danych z całego świata. Nagrodą główną w konkursie Netflix Prize był 1 milion dolarów, a wyzwanie polegało na poprawie dokładności rekomendacji o co najmniej 10% w stosunku do wykorzystywanego wtedy systemu Cinematch.

Na potrzebę konkursu firma udostępniła uczestnikom zbiór danych, zawierający ponad 100 milionów ocen filmów wystawionych przez ponad 480 tysięcy losowo wybranych użytkowników dla blisko 18 tysięcy filmów. Jak podaje Michael Schrage, był to największy zestaw danych udostępniony publicznie. Dane zostały zebrane między 1999 i 2005 oraz odzwierciedlały rozkład wszystkich ocen otrzymanych przez Netflix w tym okresie. Podzbiór udostępniony przez Netflix został wybrany losowo spośród ocen użytkowników z minimum 18 ocenami. Działanie poszczególnych algorytmów zestawiono względem zestawu danych konkursowych liczącego 3 miliony ocen. Porównywaną statystyką był pierwiastek średniej kwadratowej błędów (ang. *Root Mean Squared Error*) [59].

Do czerwca 2007 roku do konkursu zapisało się ponad 20 tysięcy zespołów ze 152 krajów. Spośród nich, 2 tysiące zespołów zgłosiło przewidywane wyniki. Ponad 650 zespołów przekroczyło wyniki Cinematch. 90 z zespołów przekroczyło 5% poprawę wyników [64]. Po pierwszym roku trwania konkursu prowadził zespół BellKor, założony przez pracowników z firmy AT&T, jednego z najstarszych przedsiębiorstw telekomunikacyjnych, założonego w 1885 roku w wyniku przekształceń firmy Bell Telephone Company [77]. Netflix nagrodził zespół BellKor nagrodą dodatkową Progress Prize (pl. *Nagrodą Postępu*) [105]. W kolejnych latach zespół BellKor dogadał się z innymi uczestnikami konkursu, tworząc BellKor's Pragmatic Chaos. Konkurs zakończył się w 2009 roku. Łącznie w konkursie udział wzięło ponad 50 tysięcy uczestników. Wspólnymi siłami, połączony zespół BellKor's Pragmatic Chaos przekroczył wymagane minimum 10% poprawy względem Cinematch i wygrał milion dolarów.

Zwycięski algorytm zasłynął jako model łączony (ang. *ensemble model*), który połączył inne algorytmy w celu stworzenia jednego, złożonego modelu. Algorytm wykorzystywał zbiór q modeli, gdzie dla każdego modelu $k = 1, 2..., q$ posiadającego macierz prognoz R_k o wymiarach $m \times n$ zawiera prognozy dla m użytkowników i n filmów. Finalna prognoza użytkownika u dla filmu i, oznaczona jako r_{ui}, była wynikiem kombinacji prognoz z każdego modelu dla tego użytkownika i filmu. Na końcu, algorytm łączył prognozy różnych modeli $R_{1,2...,q}$ w jedną końcową prognozę. Firma Netflix nie wykorzystała jednak zwycięskiego algorytmu z uwagi na wysokie koszty uruchomienia algorytmu w środowisku produkcyjnym i przekształcenie w usługę streamingową, która miało miejsce wkrótce potem [52].

Zaproponowany model łączony składał się z kilku przedstawionych poniżej składowych. Ich dokładniejszy opis znajduje się w dalszej części niniejszej pracy.

- Model prognozy bazowej
- Model SVD w wydaniu timeSVD++
- Model sąsiedztwa z dynamiką czasową
- Ograniczone maszyny Boltzmanna

- Drzewa decyzyjne wspomagane gradientem

Wszystkie modele wykonywały operacje na rzadkiej, wspomnianej wcześniej macierzy $m \times n$. Rzadka macierz (ang. *sparse matrix*) charakteryzuje się niewielką ilością przypisanych wartości. W przypadku konkursu Netflix Prize, udostępnione oceny stanowiły zaledwie około 1.2% całkowitej ilości kombinacji użytkownik-film [95]. Wynika to z faktu, że większość użytkowników nie oceniała większości filmów, a co za tym idzie, tylko niewielki ułamek potencjalnych ocen istnieje. W rezultacie, macierz ta była wysoce rzadka, co stanowiło wyzwanie w kontekście opracowywania i wdrażania algorytmów rekomendacji.

3.3 Amazon

Amazon to amerykańska firma technologiczna, która obecnie skupia się na trzech sektorach: handlu online (ang. *e-commerce*), technologii obliczeń w chmurze poprzez *Amazon Web Services* [14] oraz sektorze usług strumieniowania cyfrowego *Amazon Prime*, *Twitch* i *Audible*. Jest uznawana za jednego z członków tzw. *Big Five* - pięciu największych amerykańskich firm technologicznych.

Platforma *Amazon.com*, specjalizująca się w *e-commerce*, wykorzystuje algorytmy rekomendacji do personalizacji sklepu dla każdego klienta. Dzięki temu może ona prezentować produkty dostosowane do zainteresowań i bieżących potrzeb odwiedzających. Skuteczność tych personalizacji jest mierzona ilością kliknięć oraz współczynnikiem konwersji mierzoną jako ilość zakupów przypadająca na ilość odwiedzin w kampaniach webowych oraz mailowych. Jak informowali pracownicy Amazon tego typu kampanie swoją skutecznością przewyższają wyniki kampanii niespersonalizowanych [67].

Na potrzeby wczesnej platformy sprzedażowej w Amazon opracowano algorytm *item-to-item collaborative filtering*, który został uruchomiony w 1998 roku [98]. Algorytm ten analizuje poprzednie zakupy i oceny, dopasowując produkty do potrzeb użytkowników przez wyszukiwanie pozycji często kupowanych razem, co jest jego główną zaletą w kontekście skalowania w obliczu ogromnej bazy danych klientów i produktów.

Kod 3.2: Pseudokod wczesnego algorytmu Amazon

```
dla każdego produktu I1:
    dla każdego klienta C:
        jeżeli kupiono I1 oraz kupiono I2, odnotuj zakup
    dla każdego produktu I2:
        oblicz podobieństwo między I1 i I2
```

Algorytm 3.2 przeszukuje dla każdego produktu w katalogu I_1, każdego klienta C, a następnie dla C przeszukuje każdy produkt I_2 tak, aby odnaleźć parę produktów I_1 oraz I_2 kupowaną częściej niż inne. Z uwagi na złożoność obliczeniową algorytmu w pesymistycznym przypadku $O(N^2M)$, stosowano w praktyce próbkowanie (ang. *sampling*), czyli uwzględnienie losowego fragmentu zestawu danych treningowych z pełnego zbioru. Dodatkowo, większość użytkowników miała niewiele produktów kupionych i ocenionych, co też redukowało złożoność. Ważnym aspektem tego modelu był fakt, że tabela powstawała offline, by następnie można było w niej wyszukiwać produkty w czasie rzeczywistym.

W kolejnych latach algorytm został zmodernizowany oraz dopasowany do nowych potrzeb zmieniającego się profilu klientów i sprzedawców platformy. Wśród rozszerzeń możliwości algorytmu znajdowały się integracje z innymi współcześnie stosowanymi technikami uczenia maszynowego i sztucznej inteligencji [63]. Wprowadzono także elementy przetwarzania języka naturalnego (ang. *natural language processing*), dzięki czemu można było przeprowadzać znacznie dokładniejsze analizy recenzji oraz opisów produktów. Uwzględniono także upływ czasu, który ma wpływ na wybory klientów oraz zoptymalizowano algorytm pod kątem przetwarzania rosnących zbiorów danych w czasie rzeczywistym. W obecnej

formie system ten, podobnie jak wiele innych wspomnianych w niniejszej pracy, jest w rzeczywistości systemem hybrydowym łączącym wiele technik i algorytmów [98].

Po ponad dwóch dekadach od uruchomienia systemu, rekomendacje stanowią dzisiaj istotną część ruchu generowaną na platformie Amazon.com. Wynikiem działania systemu jest blisko 30% ruchu pochodzącego z rekomendacji [97]. Rekomendacje są serwowane milionom użytkowników spośród milionów produktów.

3.4 Spotify

Spotify to platforma streamingowa, która udostępnia bazę utworów muzycznych i podcastów na żądanie. Została założona w 2006 roku w Sztokholmie, w Szwecji przez Daniela Ek oraz Martina Lorentzona. Platforma posiada w swojej kolekcji ponad 100 milionów utworów od ponad 11 milionów artystów. Z racji ogromnej ilości danych, istotnym dla efektywnego działania platformy było opracowanie skutecznego mechanizmu sugerowania treści, które mogą zainteresować użytkownika [35]. Utwory oraz sugestie są udostępniane w kilku modułach:

- Odkryj - playlisty z proponowanymi utworami, personalne, grupowane oraz regionalne
- Odkryj w tym tygodniu
- Polecane teraz
- Radio - na bieżąco generowane rekomendacje
- Powiązani artyści - sekcja w profilu artysty, sugerująca innych, powiązanych

W tym celu stworzono system *Bandits for Recommendations as Treatments, BaRT*, który wykorzystuje mieszany mechanizm jednorękiego bandyty i filtrowania kolaboracyjne [103] [95].

Aplikacja Spotify nie posiada opcji oceniania utworów w skali przykładowo 1-5. Zamiast tego informacje zwrotne są zbierane w oparciu o zachowanie użytkowników. Gdy użytkownik jawnie doda utwór do ulubionych lub gdy spędził minimum 30 sekund słuchając, BaRT oznaczy go jako interesujący.

Zebrane dane są analizowane i przetwarzane przy pomocy Hadoopa i Cassandry, a następnie wyszukiwane jest podobieństwo na podstawie algorytmu przybliżonych najbliższych sąsiadów. Dane o utworach i użytkownikach są reprezentowane za pomocą wektorów. Ze względu na złożoność algorytmu i skalę działania Spotify, tradycyjne algorytmy k-najbliższych sąsiadów były zbyt wolne. Z tego powodu Erik Bernhardsson opracował bibliotekę *Annoy* (ang. Approximate Nearest Neighbors Oh Yeah) w C++ [34].

Annoy pozwala na odnalezienie k-punktów w przestrzeni wektorowej, potencjalnie najbliższych dla wskazanego punktu. Annoy wspiera następujące funkcje agregujące do obliczania odległości:

- Odległość euklidesową
- Odległość Manhattan
- Odległość cosinusową
- Odległość Hamminga
- Odległość iloczynu skalarnego

Inną istotną funkcją, którą oferuje jest możliwość zapisania i odczytania plików *mmap, ang. memory map*. Dzięki temu można raz zbudować indeks, aby następnie tak przygotowane dane przeszukiwać. Z uwagi na złożoność obliczeniową algorytmu, polecane w ramach Odkryj są obliczane z wyprzedzeniem. Przykładowo playlista *Odkryj w tym tygodniu* przechowywana jest przez tydzień.

W celu zoptymalizowania tworzenia playlist tworzony jest vector uśredniony z polubionych utworów. Reprezentuje on *gust muzyczny*. Wektor ten może zostać następnie użyty do wyszukania podobnych utworów wykorzystując Annoy.

Kolejna składowa BaRT opiera się o przetwarzanie języka naturalnego *ang. natural language processing, NLP*. Z tytułu oraz treści utworów tworzone są wektory osadzeń, które reprezentują w postaci numerycznej treść [95]. Następnie porównywana jest semantyczne i znaczeniowe podobieństwo między utworami. Jeżeli wyrazy znajdują się w treści utworów znajdujących się w jednej playliście, ich wektory powinny znaleźć się bliżej siebie. W dalszej części pracy opisano obliczanie osadzeń słów oraz podobieństwa z wykorzystaniem algorytmów Word2Vec i k-najbliższych sąsiadów.

Jednakże, bywa, że pojedyncze utwory znacząco różnią się od ogólnego gustu muzycznego słuchacza. Odpowiednio dokładne wykrycie takiego odchylenia może pozwolić na modyfikację wyników lub zignorowanie niestandardowych rekomendacji, aby nie zaburzyć ogólnego działania algorytmu.

3.5 Reddit

Reddit to wielotematyczne forum dyskusyjne, którego struktura opiera się na koncepcji *subredditów*, czyli kategorii [106]. Użytkownicy mogą dodawać, komentować i głosować na treści. Za aktywność na platformie użytkownicy zbierają punkty *karmy*.

Rekomendacje treści na Reddit są tworzone w oparciu o profile użytkowników. Polecane są im wątki z kategorii, które mogą wydać się interesujące. Firma nie ujawnia publicznie, w jaki sposób tworzone są rekomendacje. Użytkownicy platformy stworzyli jednak wiele własnych systemów rekomendacji, w tym RedTweet [78].

System RedTweet działa na zasadzie analizy danych z Twittera, gdzie użytkownicy publikują krótkie wiadomości, znane jako tweety. RedTweet przetwarza te tweety, aby określić preferencje tematyczne użytkowników. Wykorzystując algorytm klasyfikacji oparty na metodzie bayesowskiej, system przypisuje tweetom kategorie odpowiadające różnym subredditom.

Wstępne testy systemu RedTweet przeprowadzono na grupie kontrolnej użytkowników, których zainteresowania były dobrze znane i dokumentowane. System osiągnął wysoką precyzję w identyfikacji zainteresowań użytkowników i rekomendacji adekwatnych treści.

System RedTweet składa się z komponentów:

1. Zbieranie danych - automatyczne gromadzenie tweetów i danych z Reddit za pomocą API. Dane te są następnie przetwarzane, aby uzyskać jasny obraz zainteresowań użytkownika.

2. Analiza i przetwarzanie tekstu - wykorzystanie narzędzi NLP do ekstrakcji i analizy gatunkowej treści tweetów. Zastosowanie algorytmów takich jak TF-IDF i Naive Bayesian w celu identyfikacji kluczowych tematów wpisu.

3. Profilowanie zainteresowań - tworzenie profili użytkowników na podstawie ich aktywności na Twitterze. Profil ten jest aktualizowany w czasie rzeczywistym, co pozwala na dynamiczne dostosowywanie rekomendacji.

4. Rekomendacje treści - algorytmy wykorzystują profile zainteresowań do sugerowania odpowiednich subredditów i wątków na Reddit, które odpowiadają zainteresowaniom użytkownika.

Ostateczne rekomendacje są generowane na podstawie *profilu zainteresowań* użytkownika, który jest dynamicznie aktualizowany. System bierze pod uwagę nie tylko dominujące kategorie w tweetach użytkownika, ale również ich częstotliwość i świeżość, co pozwala na dostosowanie rekomendacji do bieżących zainteresowań.

3.6 YouTube

YouTube, popularna platforma wideo należąca do giganta technologicznego Google, jest serwisem umożliwiającym twórcom wideo dzielenie się swoimi treściami. Jak podkreśla Crisos Goodrow, pełniący funkcję wiceprezesa ds. inżynierii w Google, *Rekomendacje generują znaczącą część wyświetleń na YouTube, przewyższając subskrypcje i wyniki wyszukiwania* [44]. Fakt ten świadczy o istotnej roli, jaką odgrywają systemy rekomendacyjne w dystrybucji i promocji materiałów wideo na tej platformie.

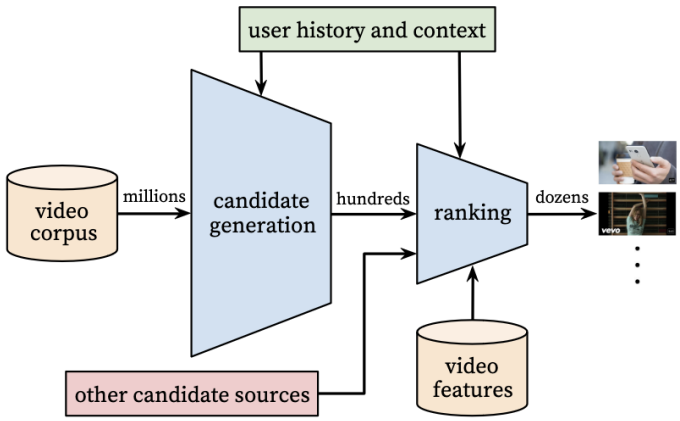
Rekomendacje na YouTube są eksponowane na stronie głównej oraz w bocznym panelu, gdzie użytkownicy mogą dodawać filmy do sekcji *Obejrzyj później* lub przechodzić do kolejnego filmu. Wcześniejsze wersje systemu ograniczały się do wyświetlania popularnych trendów, obecnie jednak system bazuje na algorytmach personalizujących, z naciskiem na filtrację kolaboracyjną.

Algorytm personalizujący rekomendacje na YouTube wykorzystuje między innymi:

1. Liczbę kliknięć na treści.
2. Czas spędzony na oglądaniu filmu.
3. Dane z ankiet użytkowników.
4. Oceny i udostępnienia.

Ogólny proces generowania rekomendacji przedstawiono na rysunku 3.2. W trakcie tego procesu ograniczana jest widoczność treści niskiej jakości, rasistowskich, pełnych przemocy oraz tych naruszających zasady platformy.

Proces rekomendacji na YouTube odbywa się w kilku etapach, wykorzystując TensorFlow [11] stworzony przez Google Brain [29]. Na początku wybierane są potencjalne filmy do polecenia spośród całej dostępnej bazy. Model kandydatów używa głębokich sieci neuronowych do transformacji danych w wektory (osadzenia), które reprezentują zarówno użytkowników, jak i filmy. System na tej podstawie ocenia prawdopodobieństwo zainteresowania użytkownika każdym filmem.



Rysunek 3.2: Architektura systemu rekomendacji YouTube. Źródło: [26]

Następnie wybrane filmy są oceniane bardziej szczegółowo. Model rankingowy przydziela każdemu filmowi wynik na podstawie takich czynników, jak prawdopodobieństwo kliknięcia (*click-through rate, CTR*) i przewidywany czas oglądania. Ranking uwzględnia nie tylko interakcje użytkownika z danym wideo, ale także z treściami pokrewnymi oraz zachowania innych użytkowników o podobnych profilach. Celem jest maksymalizacja zaangażowania użytkownika poprzez rekomendowanie treści, które chętnie będzie oglądał [26].

Po wytrenowaniu, wyniki z modelu rankingowego są poddawane post-processingowi za pomocą reguł biznesowych (np. dodatkowe filtrowanie treści nieodpowiednich) i algorytmów optymalizacji, które dostosowują listę rekomendowanych filmów do ostatecznych celów biznesowych, takich jak zwiększenie zaangażowania użytkowników.

3.7 TikTok

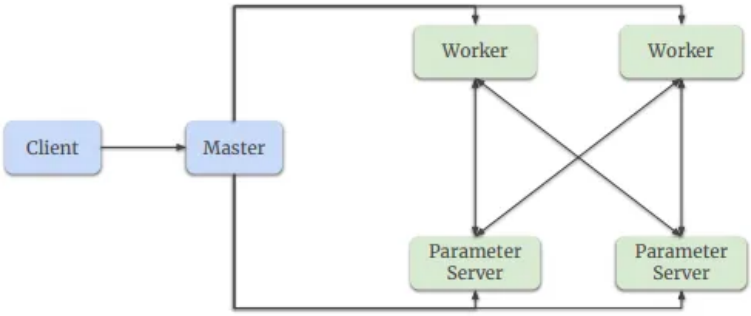
TikTok to popularna aplikacja i platforma mediów społecznościowych do tworzenia i udostępniania krótkich filmów wideo, zazwyczaj o długości od 15 sekund do trzech minut. Powstała ona we wrześniu 2016 roku w Chinach pod nazwą *Douyin*. ByteDance, założona w 2012 roku przez Zhanga Yiminga, jest chińską firmą technologiczną, która jest również właścicielem innych aplikacji, takich jak platformy newsowej Toutiao. W 2017 roku ByteDance wprowadziło na rynek międzynarodową wersję Douyin pod nazwą TikTok, dostosowując aplikację do globalnej publiczności. Aplikacja została utworzona w 200 dni i szybko zyskała rozgłos [55]. W 2018 roku aplikacja stała się najczęściej pobieraną na świecie, osiągając ponad 104 miliony pobrań. Użytkownicy w Chinach spędzali na platformie średnio 75 minut dziennie, więcej niż na Facebooku i prawie dwukrotnie więcej niż na Snapchacie. Łącznie platforma miała wówczas ponad 120 milionów aktywnych użytkowników globalnie.

Od swojego startu aplikacja śledziła poczynania użytkowników. Zbierała informacje na temat poszczególnych kliknięć, przewinięć rolek, czasu spędzonego na filmikach i lokalizacji by wychwycić regionalne trendy. System śledzenia w TikToku różni się od tego znanego z Facebooka z uwagi na brak konieczności wykonywania jawnych akcji takich jak polubienie artykułu lub napisanie komentarza [109]. Nie ma też powiązań między treściami jak ma to miejsce w grafach stanowiących podstawę platformy Facebook.

Jak wspominała Wei-Ying Ma: *Wyciągnęliśmy setki wysoko wymiarowych cech użytkowników i przygotowaliśmy model zainteresowań użytkownika bazując na tych danych* [95].

Większość systemów rekomendacji musi mierzyć się z problemem zimnego startu (ang. *cold start problem*). Problem ten odnosi się do trudności w generowaniu odpowiednich rekomendacji dla nowych użytkowników lub nowych elementów, dla których nie ma wystarczających danych historycznych [61]. Bez wystarczającej historii aktywności, systemowi trudno jest przewidzieć, co może spodobać się użytkownikowi lub jak nowy element zostanie odebrany przez innych użytkowników. Prowadzi to do sytuacji, w której jakość rekomendacji jest niska do czasu zebrania wystarczającej ilości danych. Podczas pierwszego uruchomienia TikTok rozwiązuje problem zimnego startu między innymi przez dobór treści zależnych od systemu operacyjnego, języka i lokalizacji użytkownika.

System opracowany przez firmę ByteDance, właściciela TikToka, nosi nazwę *Monolith* [68]. Działanie systemu opiera się na architekturze opartej na treningu w czasie rzeczywistym, umożliwiającym ciągłe aktualizacje parametrów modelu w oparciu o zachowania użytkowników. Monolith wykorzystuje architekturę *Worker-Parameter Server*, znaną z frameworka do uczenia maszynowego - TensorFlow. Diagram przedstawiający architekturę Worker-Parameter Server przedstawiono na rysunku 3.3. Architektura ta pozwala na zrównoleglenie procesu trenowania modelu, co przyspiesza jego aktualizację i dostosowanie do nowych danych [68].



Rysunek 3.3: Architektura Worker-Parameter Server. Źródło: [68]

Monolith składa się z modułów:

1. Parameters Server, który przechowuje parametry modelu, w tym zarówno gęste (ang. *dense*), jak i rzadkie (ang. *sparse*) parametry. Serwer ten odpowiada za aktualizację parametrów na podstawie gradientów obliczonych przez maszyny robocze (ang. *workers*).
2. Maszyny robocze, których zadaniem jest wykonywanie obliczeń związanych z przetwarzaniem danych wejściowych oraz backpropagation w celu obliczenia gradientów, które są następnie wysyłane do serwera parametrów.
3. Streaming Engine, przy pomocy *Apache Fink* przetwarza strumieniowo dane użytkowników i ich działania. W ten sposób tworzone są dane treningowe na bieżąco.
4. Hash Table z Cuckoo Hashing, struktura danych, która eliminuje problemy związane z kolizjami i pozwala na efektywniejsze zarządzanie rozległymi zbiorami.
5. Parameter Synchronization, zapewnia, że każda aktualizacja modelu jest natychmiast odzwierciedlana w serwowanych użytkownikom treściach.

Monolith, dzięki swojej architekturze, jest jednym z modelowych przykładów efektywnego wykorzystania danych w czasie rzeczywistym do personalizacji treści.

4. Analiza metod rekomendowania treści

Poniższy rozdział omawia metody rekomendacji, zaczynając od filtrowania opartego na treści, poprzez techniki filtrowania kolaboracyjnego, a kończąc na maszynach Boltzmanna. Przedstawiono algorytmy takie jak metoda bazowa BellKor, kluczowa w przetwarzaniu języka naturalnego metoda TF-IDF oraz techniki takie jak osadzanie słów i klasteryzacja K-Means. Następnie opisano metody filtrowania oparte o współpracę, w tym rozkład macierzy SVD, model sąsiedztwa z dynamiką czasową, który adaptuje oceny użytkowników w zależności od zmieniających się preferencji czasowych. Wymieniono również metody takie jak LDA i PCA, używane do grupowania dokumentów i redukcji wymiarów. Na zakończenie, zaprezentowano biblioteki i pakiety w środowisku Python, takie jak Sci-Kit Learn, Surprise i NLTK, wspierające implementację i stosowanie omówionych metod rekomendacji.

4.1 Filtrowanie oparte na treści

Filtrowanie oparte na treści to technika stosowana w systemach rekomendacji, która koncentruje się na analizie atrybutów takich jak treść opisu, metadane i specyficzne cechy [92]. Polega ona na tworzeniu szczegółowych profili przedmiotów podlegających rekomendacji na podstawie dostępnych danych, które mogą obejmować opis tekstowy, autora, rok publikacji, gatunek, a nawet specyfikacje techniczne produktu. Proces ten koncentruje się na wykorzystaniu informacji bezpośrednio związanych z przedmiotami, bez konieczności analizowania interakcji użytkownika.

4.1.1 Prognoza bazowa BellKor

Wśród wielu dostępnych metod i modeli ten zaproponowany przez zespół BellKor w trakcie konkursu Netflix Prize [99] jest jednym z popularniejszych. W przypadku modelu BellKor prognoza bazowa b_{ui} dla oceny r_{ui} obliczana była tak, aby uwzględnić ogólne tendencje użytkowników [59]. Wstępna prognoza bazowa estymowała, na ile ocena użytkownika różni się od średniej, biorąc pod uwagę ogólną średnią ocen oraz średnią ocen użytkownika. Równanie:

$$b_{ui} = \mu + b_u + b_i \tag{4.1}$$

gdzie b_u i b_i odpowiadały odchyleniom użytkownika i filmu od średniej. Na przykład, film Titanic wydany w 1997 w reżyserii Jamesa Camerona, ma średnią ocenę 4,2. Średnia wszystkich ocen wynosi 3,7. Użytkownik ocenia filmy 0,3 poniżej średniej. Wynik dla tego filmu wynosi 3,7-0,3+(4,2-3,7)=3,9. W praktyce stosowano bardziej złożone metody szacowania parametrów dla prognoz bazowych [59].

Dla każdego filmu i:

$$b_i = \frac{\sum_{u \in R(i)} (r_{ui} - \mu)}{\lambda_1 + |R(i)|} \tag{4.2}$$

oraz dla każdego użytkownika u:

$$b_u = \frac{\sum_{i \in R(u)} (r_{ui} - \mu - b_i)}{\lambda_2 + |R(u)|} \tag{4.3}$$

gdzie:

- $R(i)$ - zbiór wszystkich użytkowników, którzy ocenili film i.
- $R(u)$ - zbiór wszystkich filmów ocenionych przez użytkownika u.
- r_{ui} - ocena.

- μ - średnia wszystkich ocen.
- λ_1, λ_2 - zewnętrzne parametry obliczone drogą walidacji.

Zaproponowano także dokładniejsze szacowanie b_u i b_i poprzez rozwiązanie:

$$\min_{b^*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_3 \left(\sum_u b_u^2 + \sum_i b_i^2 \right) \tag{4.4}$$

gdzie:

- b^* - wszystkie odchylenia użytkowników i filmów.
- λ_3 - zewnętrzny parametr obliczony drogą walidacji.
- $\mathcal{K}$ - zbiór wszystkich par użytkownik-filmów z ocenami.

Do prognozy bazowej BellKor zaproponował modyfikacje, które zmniejszają RMSE. Jedną z nich było zauważenie, że oceny użytkowników mogą zmieniać się w czasie. W związku z tym składowe potraktowano jako funkcje w czasie.

$$b_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}) \tag{4.5}$$

BellKor zwrócił uwagę, że popularność filmów ewoluuje stopniowo i nie zmienia się z dnia na dzień. Natomiast zachowania użytkowników mogą wykazywać większą zmienność, nawet na co dzień. Do modelowania zmienności czasowej $b_i(t)$, bias filmu podzielono na segmenty czasowe. Umożliwiło to lepsze dopasowanie modelu do danych zależnych od okresów takich jak dekady czy sezony. Modelowanie skłonności użytkowników z uwzględnieniem zarówno długoterminowych trendów, jak i krótkoterminowych zmian pozwoliło na lepsze zrozumienie i przewidywanie zachowań oceniających.

Ponadto, BellKor wykazał, że zmienność danych może być również wyjaśniona przez częstotliwość, z jaką użytkownicy wystawiają oceny w określonym dniu. To zjawisko zostało uwzględnione w modelu poprzez wprowadzenie parametru częstotliwości, który modyfikuje bias przedmiotu w zależności od logarytmu zaokrąglonej liczby ocen wystawionych przez użytkownika w danym dniu.

Zastosowanie dodatkowego czynnika znacząco obniżało RMSE, co wskazuje na to, że częstotliwość wystawiania ocen miała istotny wpływ na jakość prognozy. W rezultacie, nawet bez uwzględnienia interakcji między użytkownikami i przedmiotami, taki rozbudowany predyktor bazowy był w stanie wyjaśnić prawie tyle samo zmienności danych, co komercyjny system rekomendacji Netflix, Cinematch.

Modele BellKor zawierały parametry specyficzne dla poszczególnych dni, które służyły do oczyszczania danych z krótkotrwałych fluktuacji, pozwalając na lepsze uchwycenie długoterminowych trendów w ocenach. W przypadku prognozowania na nieznane przyszłe daty, parametry te przyjmowały domyślne wartości, co pozwalało na zachowanie spójności modelu. Chociaż parametry te są niezbędne do poprawnego modelowania przeszłości, nie są one wykorzystywane bezpośrednio do przewidywania przyszłych ocen, ponieważ przewidywanie koncentruje się na wyodrębnieniu bardziej trwałych charakterystyk zachowań użytkowników.

W finalnym modelu BellKor wykorzystano technikę spadku gradientu stochastycznego do optymalizacji parametrów, co pozwoliło uzyskać wartość RMSE równą 0.9555. Algorytm spadku gradientu stochastycznego jest metodą optymalizacji, która minimalizuje funkcję kosztu przez iteracyjne dostosowywanie parametrów modelu. W każdej iteracji wybiera losowy podzbiór danych do obliczenia gradientu, co czyni proces szybszym i bardziej efektywnym. Dodatkowo, do modelu włączono bardziej precyzyjne prognozy bazowe, co pozwoliło dalej obniżyć RMSE do 0.9278.

Ostatecznie wybrane wartości parametrów po 40 iteracjach przedstawiono w tabeli 4.1.

Tabela 4.1: Wartości współczynników dla poszczególnych parametrów modelu BellKor [59]

	b_u	b_{ut}	α_u	b_i	$b_{i,Bin(t)}$	c_u	c_{ut}	$b_{i,f_{ui}}$
lrate ($\times 10^3$)	2.67	2.57	3.11e-3	0.488	0.115	5.64	1.03	2.36
reg ($\times 10^2$)	2.55	0.231	395	2.55	9.29	4.76	1.90	1.10e-6

4.1.2 TF-IDF

Częstotliwość terminu - odwrotna częstotliwość dokumentu (ang. *term frequency-inverse document frequency (TF-IDF)*) jest metodą przetwarzania języka naturalnego, wykorzystywaną w systemach rekomendacji treści oraz klasyfikacji dokumentów [16]. Model ten opiera się na założeniu, że znaczenie słowa w dokumencie rośnie wraz ze wzrostem jego częstotliwości, przy jednoczesnym rzadkim występowaniu w całym zbiorze dokumentów (po usunięciu słów pustych, ang. *stop words*). Pierwszy komponent metody, częstotliwość terminu (ang. *term frequency*), może być obliczana na różne sposoby. Kim Falk w publikacji *Practical Recommender Systems* [37] sugeruje następujący wariant:

$$tf(t,d) = 1 + \log(f_{t,d}) \tag{4.6}$$

gdzie:

- t - termin (wyraz lub fraza), którego częstotliwość jest mierzona.
- d - dokument, w którym mierzona jest częstotliwość.
- $f_{t,d}$ - liczba wystąpień terminu t w dokumencie d.

By uzyskać precyzyjne wyniki, poza słowami pustymi należy odfiltrować słowa, które dodają kontekstu lub znaczenia w niewielkim stopniu. Istnieją różne metody czyszczenia tekstu. Jedną z nich jest stemizacja opisana w dalszej części niniejszej pracy, inną przeszukiwanie z użyciem wyrażeń regularnych.

W tym celu można zastosować odwróconą częstotliwość dokumentu (ang. *inverse document frequency*). Podobnie jak w przypadku tf istnieją różne warianty sposobu liczenia. Zazwyczaj równanie opisujące idf wygląda następująco:

$$idf(t,D) = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right) \tag{4.7}$$

gdzie:

- t - termin do wyszukania.
- D - zbiór dokumentów do przeszukania.
- N - rozmiar zbioru dokumentów $N = |D|$.
- $|\{d \in D : t \in d\}|$ - rozmiar zbioru dokumentów zawierających termin t.

Cały model definiuje się jako:

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D) \tag{4.8}$$

gdzie:

- t - termin do wyszukania.
- d - dokument, w którym mierzona jest częstotliwość.

- D - zbiór dokumentów do przeszukania.

W kodzie 4.1 zamieszczono przykład obliczenia tf-idf dla dokumentów zawierających przykładowe opisy filmów z superbohaterami i frazy *thanos*. Zakładamy, że każdy z tych opisów jest osobnym dokumentem.

Opis filmu Avengers

> *Film Avengers z 2012 roku opowiada o grupie superbohaterów, która musi zapomnieć o indywidualnych różnicach i połączyć siły, by powstrzymać Lokiego, zdradzieckiego brata Thora, który chce podbić Ziemię przy pomocy armii kosmitów. Superbohaterowie, w tym Iron Man, Kapitan Ameryka, Thor, Hulk, Czarna Wdowa i Sokółi Okiem, muszą nauczyć się współpracować jako zespół, by stawić czoła globalnemu zagrożeniu. W trakcie filmu, drużyna przechodzi transformację z grupy indywidualistów w zgrany zespół, który łączy siły, by uratować świat.*

Opis filmu Avengers: Infinity War

> *Avengers: Infinity War przedstawia historię Thanosa, potężnego tyrana, który dąży do zdobycia Kamieni Nieskończoności, aby uzyskać moc sterowania rzeczywistością. Superbohaterowie z całego uniwersum Marvela, w tym Avengers i Strażnicy Galaktyki, muszą połączyć siły, by stawić czoło Thanosowi i jego armii. Film ukazuje desperacką walkę o przetrwanie i ochronę wszechświata, kończąc się dramatycznym i niespodziewanym zwrotem akcji.*

Opis filmu Avengers: Koniec gry

> *Avengers: Koniec gry kontynuuje wydarzenia po Infinity War, gdzie Thanos zniszczył połowę wszelkiego życia we wszechświecie. Pozostali przy życiu superbohaterowie, w tym Iron Man, Kapitan Ameryka, Thor, Czarna Wdowa, Hawkeye i Hulk, łączą siły, by odwrócić działania Thanosa. W trakcie filmu opracowują plan podróży w czasie, aby odzyskać Kamienie Nieskończoności z przeszłości i uratować przyszłość, co prowadzi do epickiej i emocjonalnej konfrontacji z Thanosem i jego armią.*

Opis filmu Batman: Mroczny Rycerz

> *Batman, wspierany przez porucznika Jima Gordona i prokuratora Harveya Denta, walczy z rosnącą przestępczością w Gotham City. Kiedy do miasta przybywa chaotyczny i bezwzględny Joker, stawiając miasto na krawędzi anarchii, Batman musi stawić czoła nie tylko fizycznym wyzwaniom, ale również moralnym dylematom. Film eksploruje tematy poświęcenia i dylematów etycznych, przedstawiając Batmana jako bohatera gotowego do poniesienia ogromnych kosztów osobistych, aby ochronić miasto.*

Opis filmu Avengers: Czas Ultrona

> *Avengers: Czas Ultrona opowiada o drużynie Avengers, która staje w obliczu Ultron, sztucznej inteligencji stworzonej przez Tony'ego Starka i Bruce'a Bannera, mającej na celu ochronę Ziemi. Ultron, odbiegając od swojego pierwotnego zadania, postanawia zniszczyć ludzkość, uznając ją za największe zagrożenie dla świata. Avengers muszą ponownie połączyć siły, by pokonać Ultrona i jego armię robotów, co prowadzi do globalnej bitwy i przemyśleń dotyczących odpowiedzialności i konsekwencji ich działań jako superbohaterów.*

Poszczególne opisy przekazujemy jako lista ciągów znaków. Opisy zostały poddane wcześniejszej stemizacji.

Kod 4.1: Obliczanie tf-idf

```
def calculate_tf(term, doc):
  words = clean_and_stem(doc)
  stemmed_term = simple_stem(term)
  word_count = words.count(stemmed_term)
  return 1 + log(word_count + 1) if word_count > 0 else 0

def calculate_idf(term, docs):
  stemmed_term = simple_stem(term)
  docs_with_term = sum(stemmed_term in clean_and_stem(doc) for doc in docs)
  return log(len(docs) / (docs_with_term if docs_with_term else 1))

def calculate_tfidf(term, docs):
  idf = calculate_idf(term, docs)
  tfidf_values = [calculate_tf(term, doc) * idf for doc in docs]
  return tfidf_values
```

Tabela 4.2: Wyniki obliczeń TF, IDF oraz TF-IDF dla opisów filmów i frazy *superbohater*.

Dokument z opisem	**TF**	**IDF**	**TF-IDF**
Avengers	2.0986	0.2231	0.4683
Avengers: Infinity War	1.6931	0.2231	0.3778
Avengers: Koniec gry	1.6931	0.2231	0.3778
Batman: Mroczny Rycerz	0.0	0.2231	0.0
Avengers: Czas Ultrona	1.6931	0.2231	0.3778

Tabela 4.3: Wyniki obliczeń TF, IDF oraz TF-IDF dla opisów filmów i frazy *thanos*.

Dokument z opisem	**TF**	**IDF**	**TF-IDF**
Avengers	0.0	0.9163	0.0
Avengers: Infinity War	2.0986	0.9163	1.9229
Avengers: Koniec gry	2.3863	0.9163	2.1865
Batman: Mroczny Rycerz	0.0	0.9163	0.0
Avengers: Czas Ultrona	0.0	0.9163	0.0

Tabela 4.4: Wyniki obliczeń TF, IDF oraz TF-IDF dla opisów filmów i frazy *batman*.

Dokument z opisem	**TF**	**IDF**	**TF-IDF**
Avengers	0.0	1.6094	0.0
Avengers: Infinity War	0.0	1.6094	0.0
Avengers: Koniec gry	0.0	1.6094	0.0
Batman: Mroczny Rycerz	2.3863	1.6094	3.8406
Avengers: Czas Ultrona	0.0	1.6094	0.0

Wyniki z tabel 4.2, 4.3 i 4.4 wskazują, że fraza *superbohater* jest bardziej znacząca w opisie filmu *Avengers* niż w pozostałych opisach, podczas gdy fraza *thanos* ma najwyższe znaczenie w opisach filmów *Avengers: Infinity War* i *Avengers: Koniec gry*. Fraza *batman* jest istotna tylko w opisie filmu *Batman: Mroczny Rycerz*. Wskazuje to na skuteczność modelu TF-IDF w identyfikowaniu znaczących terminów w różnych dokumentach.

4.1.3 Stemizacja

Stemizacja (ang. *stemming*) to proces redukcji słów do ich formy bazowej. Ten proces nie zawsze przekształca słowa do ich faktycznych form leksykalnych znanych z słownika. Jego głównym celem jest usunięcie afiksów - przedrostków i przyrostków [70]. Na przykład, słowa takie jak *run*, *runner*, *running* zostaną wszystkie zredukowane do rdzenia *run*. To pozwala systemom przetwarzania języka naturalnego lepiej interpretować zapytania i oferować wyniki, które są bardziej spójne bez względu na konkretną formę słowa użytego w zapytaniu. Jest szeroko stosowana w systemach przetwarzania języka naturalnego do:

- Ulepszania mechanizmów wyszukiwania informacji przez redukcję różnych form słownych do wspólnego rdzenia, co pozwala lepiej dopasować zapytania do dokumentów.
- Agregacji danych statystycznych w analizie tekstu, co jest przydatne w analizie sentymentu i klasyfikacji tekstu.

Podstawowa implementacja stemizacji przedstawiona została w kodzie 4.2. Ta implementacja usuwa sufiksy z wyrazów, redukując je do formy bazowej.

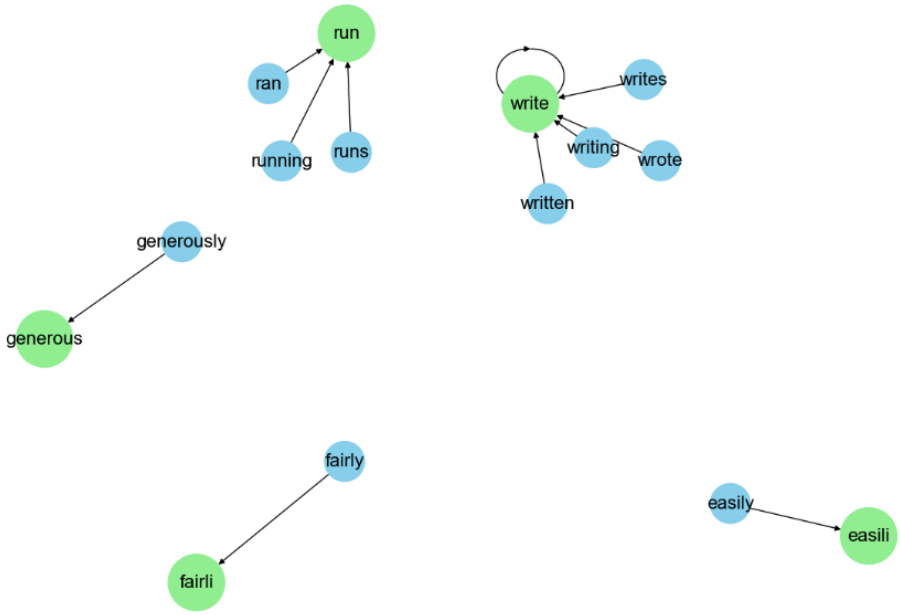
Kod 4.2: Podstawowa implementacja stemizacji

```
def naive_stem(word):
    suffixes = ['ing', 'ly', 'ed', 'ious', 'ies', 'ive', 'es', 's', 'ment']
    for suffix in suffixes:
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word
```

Martin Porter zaproponował bardziej złożony algorytm w 1980 roku [7]. Algorytm składał się z sekcji kroków i był przygotowany pierwotnie z myślą o języku angielskim. W kolejnych krokach algorytm wykonywał:

1. Usunięcie końcówek wskazujących na liczbę mnogą.
2. Przekształcenie form przeszłych do bezokoliczników.
3. Usunięcie przyrostków rzeczowników.
4. Finalne usunięcie pozostałych końcówek.

Inny algorytm zwany Snowball jest rozszerzeniem stemizacji Portera. Opracowana została przez tego samego autora w 2001 roku [6]. Snowball jest frameworkiem do pisania algorytmów stemizacji. Obsługuje wiele języków z różnych rodzin. Przykładowo romańskich: francuski, hiszpański, kataloński, portugalski i włoski. Przykładowy wynik działania i relacji między wejściowymi i wyjściowymi wyrazami pokazano na rysunku 4.1.



Rysunek 4.1: Przykładowy wynik algorytmu Snowball

Podczas gdy stemizacja koncentruje się na usunięciu końcówek słów w celu osiągnięcia formy podstawowej, lematyzacja jest procesem, który bierze pod uwagę kontekst leksykalny i morfologiczny słów, aby przekształcić je w ich kanoniczną formę słownikową. Na przykład, lematyzacja prawidłowo przekształci słowo *better* w *good*, podczas gdy stemizacja mogłaby zredukować je tylko do *better*.

Biblioteka NLTK opisana w dalszej części tego rozdziału zawiera gotowe implementacje wspomnianych algorytmów Porter i Snowball.

4.1.4 Osadzenia wektorowe słów

Osadzenia wektorowe słów (ang. *word embedding*) to metoda stosowaną w uczeniu maszynowym, pozwalającą na konwersję danych tekstowych, głównie słów, do postaci wektorów liczbowych w przestrzeni wektorowej [50]. Wektory te reprezentują semantyczne znaczenie słów w sposób, które może być efektywnie wykorzystany przez inne algorytmy. Nazywamy je osadzeniami. Choć najczęściej możemy o nich usłyszeć w kontekście osadzania słów to mogą one reprezentować inne dane. Firma Pinterest, na przykład, wykorzystuje osadzenia nie tylko dla słów, ale również dla obrazów, grafów, zapytań oraz danych użytkowników [108].

Wszystkie wektory w danej przestrzeni wektorowej są jednakowej, stałej długości, co jest wymogiem dla większości modeli uczenia maszynowego wykorzystujących te techniki. Ilość wymiarów wektora waha się w zależności od przypadku użycia i przeznaczonej mocy obliczeniowej. Proces trenowania modelu tworzy osadzenia. W wyniku uczenia zwroty o podobnych znaczeniach mają tendencję znajdować się bliżej do siebie w przestrzeni wektorów [47]. Przykładowa rodzina metod uczenia maszynowego, która pozwala stworzyć osadzenia to Word2vec. Została stworzona przez pracowników firmy Google w 2013 roku [30].

Dwie główne architektury Word2vec:

- *Continuous Bag of Words (CBOW)* - przewiduje słowo środkowe na podstawie słów otaczających to słowo. Architektura ta nie bierze pod uwagę kolejności słów w kontekście.
- *Skip-Gram* - przewiduje słowa sąsiadujące wokół docelowego, bazując wyłącznie na tym słowie.

Model CBOW używa kontekstu (słów otaczających szukane słowo) do przewidywania kolejnego słowa w danym kontekście. Jest to odwrotność modelu *skip-gram*, który z jednego słowa przewiduje jego kontekst [88]. CBOW jest szczególnie efektywny w sytuacjach, gdzie dostępne dane są duże i wymagana jest szybka i efektywna predykcja słowa na podstawie jego kontekstu.

Model jest trenowany na korpusie tekstowym, będącym zbiorem tekstów pisanych i mówionych, analizowanym w ramach przetwarzania języka naturalnego (NLP, ang. *natural language processing*). Polskie modele językowe trenowane są [36] najczęściej w oparciu o Narodowy Korpus Języka Polskiego udostępniany przez Instytut Podstaw Informatyki PAN [80]. Natomiast anglojęzyczne w oparciu o WordNet [104] udostępniany przez Princenton University lub OntoNotes [25] udostępniane przez Linguistic Data Consortium. Kolejne kroki tworzenia osadzeń polegają na wprowadzaniu tekstu do modelu, który konwertuje słowa na wektory w taki sposób, że słowa o podobnych znaczeniach są umieszczane blisko siebie w przestrzeni wektorowej. Proces ten wykorzystuje techniki takie jak redukcja wymiarowości i normalizacja wektorów, aby przyspieszyć operacje matematyczne wykonywane na tych wektorach.

Przykładowy algorytm generowania osadzeń działa następująco:

1. Na początku wszystkie wagi są inicjalizowane losowo. Wagi te są później aktualizowane podczas procesu uczenia.

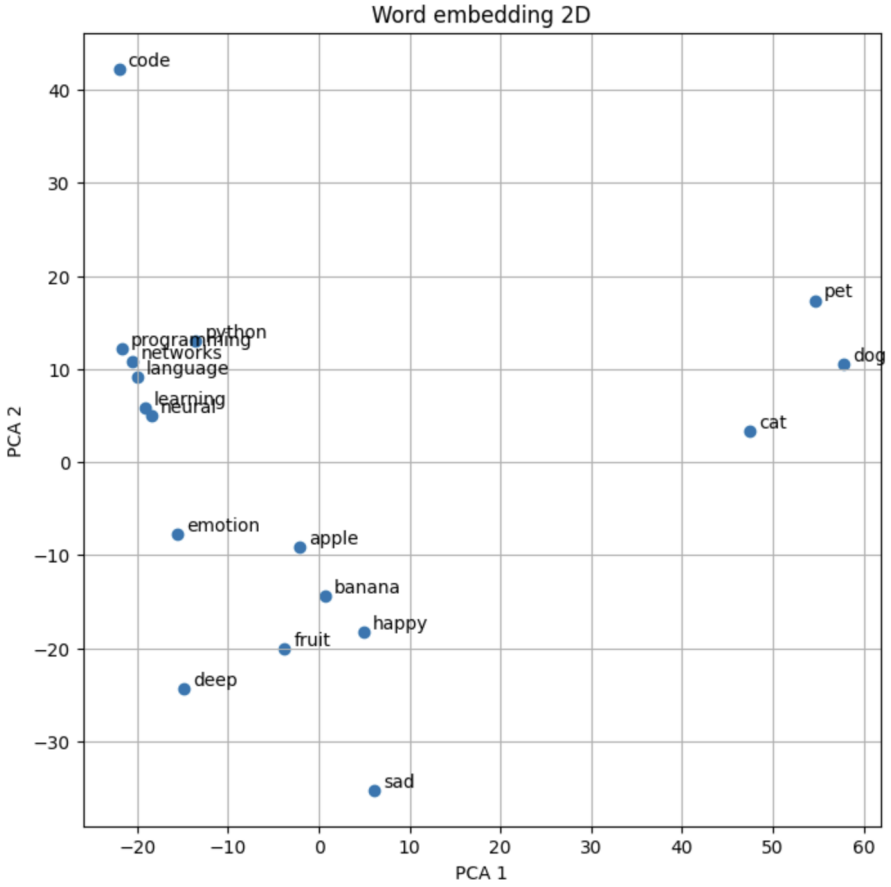
2. Dla danego słowa w korpusie, kontekst jest reprezentowany przez okno składające się z C słów przed i po danym słowie-celu. Kontekst ten jest transformowany do postaci wektorowej.

3. Obliczona średnia wektorów kontekstowych stanowi wejście dla jednowarstwowej sieci neuronowej bez warstw ukrytych, która za pomocą funkcji softmax estymuje prawdopodobieństwo docelowego słowa.

4. Wagi są aktualizowane za pomocą algorytmu optymalizacyjnego, takiego jak stochastyczny spadek wzdłuż gradientu (SGD, ang. *stochastic gradient descent*), w celu minimalizacji błędu między przewidywanym i rzeczywistym słowem.

5. Proces jest powtarzany dla kolejnych słów w korpusie do osiągnięcia zbieżności lub wyczerpania liczby epok.

Przykładowa implementacja Word2vec dostępna jest w bibliotece spaCy [72]. Wektory w modelu są wytrenowane na dużych korpusach tekstowych i mogą uchwycić semantyczne i syntaktyczne zależności między słowami [66].



Rysunek 4.2: Dwuwymiarowa reprezentacja osadzania wyrazów. Redukcja wymiarów przeprowadzona algorytmem PCA

Na rysunku 4.2 przedstawiono wynik działania modelu *en_core_web_md* i redukcję wymiarowości z 300 do 2 za pomocą PCA (ang. *principal component analysis*).

4.1.5 Klasteryzacja z wykorzystaniem k-means

Klasteryzacja (inaczej zwana segmentacją) z wykorzystaniem algorytmu k-means jest jednym z podstawowych algorytmów uczenia nienadzorowanego [37]. Metoda ta została opracowana jako technika analizy statystycznej, która pozwala na grupowanie zbioru obiektów w k klastrów, gdzie każdy obiekt należy do klastra o najbliższym średnim punkcie, zwanym centroidem [13]. Algorytm K-Means, wprowadzony przez Hugo Steinhausa w 1956 roku i rozwinięty przez Jamesa MacQueena w 1967 roku, jest jednym z najpopularniejszych metod stosowanych w analizie skupień.

Algorytm inicjuje działanie przez umieszczenie k centroidów w losowych punktach. W kolejnych iteracjach, do każdego centroidu przypisywane są najbliższe mu punkty, a następnie obliczana jest nowa pozycja centroidu jako średnia arytmetyczna współrzędnych wszystkich punktów w klastrze:

$$c_k = \frac{1}{|C_k|} \sum_{x \in C_k} x \tag{4.9}$$

gdzie:

- c_k jest centroidem klastra C_k,
- $|C_k|$ oznacza liczbę elementów w klastrze C_k,

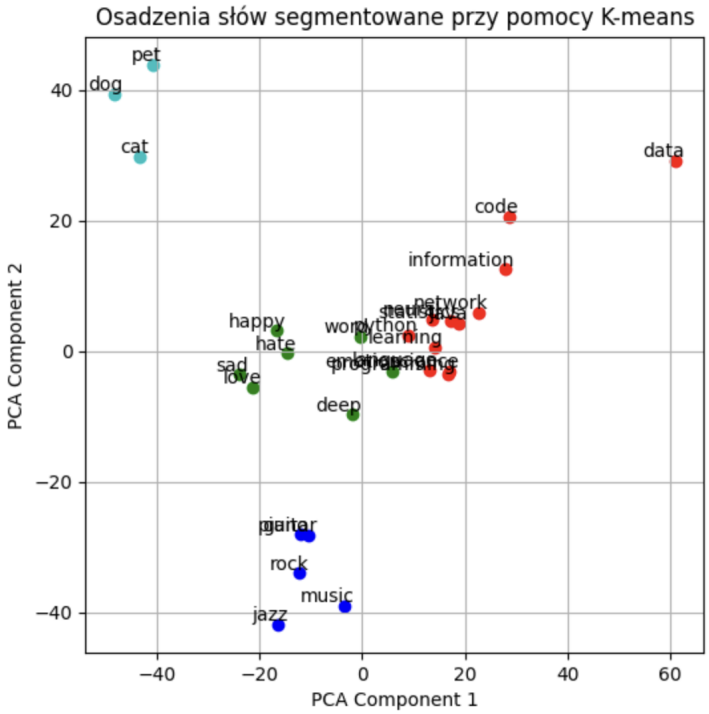
- x reprezentuje punkty należące do klastra C_k.

Dla przestrzeni wielowymiarowej c_k w każdym wymiarze j obliczany jest zgodnie z równaniem:

$$c_{kj} = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_{ij} \tag{4.10}$$

gdzie x_{ij} to j-ta współrzędna i-tego punktu w klastrze C_k.

Proces wyznaczania nowej pozycji klastra wykonywany jest, aż centroidy przestaną się przemieszczać czyli znajdą się w pozycji optymalnej (konwergentnej). Przykładowy wynik działania algorytmu k-means przedstawiono na rysunku 4.3.



Rysunek 4.3: Zredukowane osadzenia słów segmentowane przy pomocy k-means

Algorytm k-means jest stosowany w systemach rekomendacji do segmentacji zarówno treści, jak i użytkowników, opisywanych wektorami cech. Te cechy mogą obejmować metadane (np. gatunki filmów, kategorie książek) oraz historyczne zdarzenia (np. poprzednie zakupy, oglądane filmy). Użytkownicy o podobnym guście mogą otrzymywać podobne propozycje na przykład z puli wysoko ocenianych przez innych przedmiotów analizy. Dzięki grupowaniu treści w klastry, systemy rekomendacji mogą przetwarzać dane szybciej i efektywniej, zwiększając skalowalność i precyzję rekomendacji.

4.2 Filtrowanie oparte o współpracę

Filtrowanie kolaboratywne to technika stosowana w systemach rekomendacji, która opiera się na analizie interakcji między użytkownikami, w przeciwieństwie do filtrowania opartego na treści, które koncentruje się na atrybutach samych elementów [92]. W filtrowaniu kolaboracyjnym systemy próbują przewidzieć zainteresowania użytkownika na podstawie ocen i preferencji innych użytkowników o podobnych gustach lub zachowaniach.

4.2.1 Rozkład macierzy według wartości osobliwych

Rozkład macierzy to proces dekompozycji macierzy na iloczyn kilku macierzy, które w pewien sposób reprezentują pierwotną macierz, ale są łatwiejsze w manipulacji i analizie [100]. Jest to często stosowana technika w analizie danych, szczególnie w systemach rekomendacji, gdzie macierz użytkowników i produktów zostaje rozłożona w celu odkrycia ukrytych wzorców i preferencji. Jednym z popularnych przykładów faktoryzacji macierzy jest rozkład według wartości osobliwych (ang. *singular value decomposition (SVD)*). W kontekście SVD, macierz jest rozkładana na trzy inne macierze.

$$A = U\Sigma V^T \tag{4.11}$$

gdzie:

- A to oryginalna macierz ocen, gdzie każdy wiersz odpowiada użytkownikowi, każda kolumna odpowiada przedmiotowi oceny, a wartości to oceny użytkowników dla danych przedmiotów poddanych ocenom.
- U jest macierzą reprezentującą ukryte cechy użytkowników. Każdy wiersz odpowiada jednemu użytkownikowi, a każda kolumna reprezentuje jedną "ukrytą cechę"użytkownika. Wartości w tej macierzy pokazują, w jakim stopniu każdy użytkownik posiada każdą z tych cech.
- Σ jest macierzą diagonalną zawierającą ważności cech. Jest to macierz diagonalna, gdzie wartości na przekątnej wskazują, jak ważna jest każda para ukrytych cech użytkownika i przedmiotu oceny w wyjaśnianiu ocen w oryginalnej macierzy.
- V^T (transpozycja macierzy V) jest macierzą reprezentującą ukryte cechy przedmiotów poddanych ocenom. Każdy wiersz reprezentuje jedną ukrytą cechę, a każda kolumna odpowiada jednemu przedmiotowi. Wartości pokazują, w jakim stopniu każdy film posiada każdą z tych cech.

Wartości na przekątnej macierzy Σ są ułożone od największej do najmniejszej. Każda z tych wartości odpowiada jednej parze cech: jednej cesze użytkownika z macierzy U i jednej cesze przedmiotu oceny z macierzy V^T. Im większa wartość, tym ważniejsza jest dana para cech w wyjaśnianiu ocen w oryginalnej macierzy. Pary cech z dużymi wartościami w Σ reprezentują silne wzorce w preferencjach użytkowników i charakterystykach przedmiotów ocen. Pary cech z małymi wartościami reprezentują słabsze lub bardziej niszowe wzorce. Możemy wykorzystać tę informację do uproszczenia naszego modelu poprzez redukcję wymiarowości. Zamiast używać wszystkich par cech, możemy zachować tylko te najważniejsze (z największymi wartościami w Σ) i zignorować resztę. To może uczynić model szybszym i mniej skomplikowanym bez dużej utraty dokładności. Klasyczny SVD zakłada, że cechy użytkowników i cechy przedmiotów ocen są stałe w czasie. Są jednak również inne odmiany, które biorą pod uwagę, że w rzeczywistości mogą się one zmieniać w czasie.

Oprócz klasycznej faktoryzacji macierzy SVD, istnieją jej warianty. BellKor w swoim rozwiązaniu wykorzystali rozszerzenie tej techniki, które uwzględnia dynamikę czasową, znane jako timeSVD++ [59]. W tej wersji modelu, cechy użytkowników są modelowane jako zależne od czasu, co pozwala na uwzględnienie zmieniających się preferencji użytkowników i ewolucji ocen przedmiotów. Model timeSVD++ składa się z kilku komponentów:

- Predyktor bazowy zależny od czasu, który uwzględnia ogólne trendy czasowe.
- Statyczne cechy przedmiotów ocen, które nie zmieniają się z czasem.
- Cechy użytkowników zależne od czasu, które ewoluują.

Zmienność czasowa jest modelowana za pomocą funkcji, która mierzy odchylenie czasowe od średniej daty oceniania dla danego użytkownika. Ponadto, model ten jest rozszerzony o świadomość częstotliwości wystawiania ocen, co pozwala na dokładniejsze modelowanie wpływu częstotliwości oceniania przedmiotów przez użytkowników na ich percepcję.

$$\hat{r}ui = \mu + b_u(t) + b_i + q_i^T \left(p_u(t) + |N(u)|^{-\frac{1}{2}} \sum j \in N(u) y_j \right) \tag{4.12}$$

gdzie:

- $\hat{r}_{ui}$ to przewidywana ocena użytkownika u dla przedmiotu i.
- μ to średnia ocena w całym systemie.
- $b_u(t)$ to bias użytkownika u zależny od czasu t.
- b_i to bias przedmiotu i.
- q_i to wektor cech przedmiotu i.
- $p_u(t)$ to wektor cech użytkownika u zależny od czasu t.
- $N(u)$ to zbiór przedmiotów ocenionych przez użytkownika u.
- y_j to dodatkowy wektor cech przedmiotu j, który pomaga modelować wpływ częstotliwości oceniania.

Zastosowanie tych technik w timeSVD++ umożliwia stworzenie bardziej zniuansowanego i dynamicznego systemu rekomendacji, który może lepiej dostosować się do zmieniających się w czasie potrzeb i preferencji użytkowników.

4.2.2 Sąsiedztwo z dynamiką czasową

Model sąsiedztwa z dynamiką czasową to algorytm wykorzystany przez zespół BellKor w ich zwycięskim rozwiązaniu w konkursie Netflix Prize [59]. Jest to rozszerzenie tradycyjnego podejścia opartego na filtrowaniu kolaboratywnym z sąsiedztwem elementów, które bierze pod uwagę zmiany preferencji użytkowników i charakterystyk przedmiotów w czasie.

W tradycyjnym filtrowaniu kolaboratywnym system rekomendacji identyfikuje podobieństwa między użytkownikami lub przedmiotami na podstawie historii ocen. Jeśli użytkownicy A i B ocenili te same przedmioty podobnie, system uzna, że mają zbliżone preferencje. Wtedy, jeśli B wysoko ocenił przedmiot nieoceniony przez A, system zarekomenduje ten przedmiot użytkownikowi A.

Jednak to podejście zakłada, że preferencje użytkowników i charakterystyki przedmiotów nie zmieniają się w czasie, co często nie jest prawdą. Gusta użytkowników ewoluują, a popularność przedmiotów fluktuuje.

Model sąsiedztwa z dynamiką czasową rozwiązuje ten problem, wprowadzając wagi interakcji między przedmiotami, które mogą zmieniać się w zależności od czasu wystawienia oceny. Jeśli dwa przedmioty były oceniane podobnie w przeszłości, ale teraz otrzymują różne oceny, model to zauważy i zaktualizuje swoje rekomendacje.

Kluczowe elementy modelu BellKor to:

- Wykorzystanie dwóch zestawów wag item-item: jednego związanego z wartościami ocen, a drugiego biorącego pod uwagę tylko to, które przedmioty zostały ocenione, niezależnie od wartości oceny. Wagi te są automatycznie uczone z danych wraz z biasami.
- Wprowadzenie funkcji zaniku. Funkcja ta waży wpływ ocen na podstawie różnicy w czasie między nimi. Oceny oddalone w czasie mają mniejszy wpływ niż oceny bliskie sobie czasowo.

- Uwzględnienie dynamiki czasowej w bazowej predykcji (biasach) poprzez modelowanie zmian popularności przedmiotów i preferencji użytkowników w czasie.
- Uczenie parametrów modelu przez minimalizację błędu kwadratowego z regularyzacją, z użyciem stochastycznego spadku gradientu.

Dzięki tym elementom, model osiągnął RMSE na poziomie 0.8870, co było znacząco lepsze niż wcześniejsze wyniki uzyskiwane metodami sąsiedztwa i przewyższało nawet niektóre bardziej złożone podejścia hybrydowe.

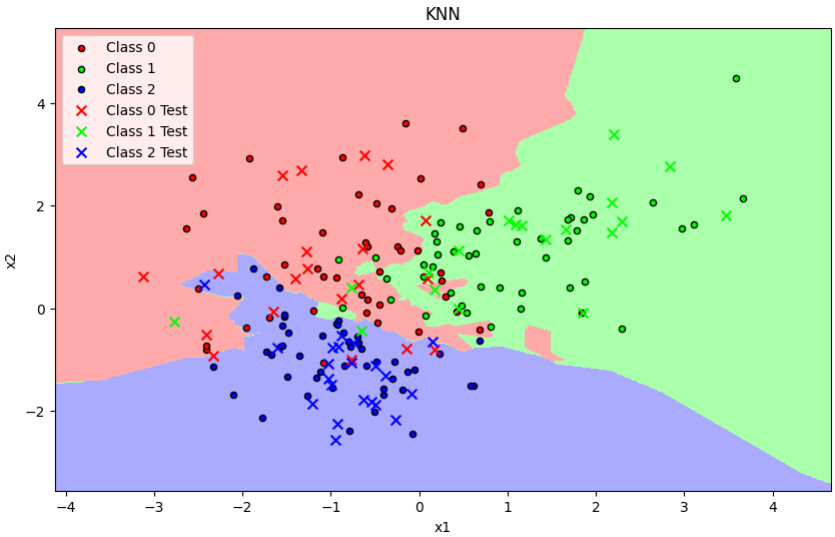
Autorzy zawarli w końcowym blendzie kilka wariantów tego modelu, różniących się szczegółami implementacji, np. liczbą iteracji uczenia (20, 25, 30), formą uwzględniania biasu (z samą dynamiką czasową lub z dodatkowymi częstościami ocen), czy eksperymentalnym wariantem z dodatkowym zestawem wag item-item d_{ij} z inną funkcją zaniku $(1 + \beta_u \Delta t)^{-1}$. Najlepszy wynik (RMSE=0.8870) uzyskano dla wariantu z 20 iteracjami uczenia i biasem uwzględniającym częstości ocen.

Kluczowy wniosek z tych eksperymentów jest taki, że odpowiednie uwzględnienie dynamiki czasowej w danych może mieć większy wpływ na dokładność niż projektowanie bardziej złożonych algorytmów uczenia. Model sąsiedztwa z dynamiką czasową pokazuje, jak stosunkowo prosta modyfikacja tradycyjnego podejścia filtrowaniu kolaboratywnym może prowadzić do znaczącej poprawy wyników.

4.2.3 K-Nearest Neighbors

Algorytm k-najbliższych sąsiadów (ang. *k-nearest neighbours*) jest jednym z podstawowych modeli generowania rekomendacji [32]. Model KNN opiera się o założenie, że elementy bliskie sobie w wielowymiarowej przestrzeni sąsiadują [83].

Algorytm klasyfikuje nowe dane poprzez głosowanie większościowe najbliższych sąsiadów w przestrzeni cech. W wersji ważonej algorytmu głosy sąsiadów mają wagę odwrotnie proporcjonalną do ich odległości od klasyfikowanego punktu, co pozwala bardziej *wpływowym* sąsiadom mieć większe znaczenie. KNN jest uznawany za *leniwy* algorytm, ponieważ nie tworzy ogólnego wewnętrznego modelu, lecz wykonuje obliczenia w momencie klasyfikacji nowych danych [94]. To odróżnia go od bardziej obliczeniowo intensywnych metod, które wymagają budowy modelu na etapie trenowania. KNN jest zatem szczególnie użyteczny, gdy potrzeba szybkiej adaptacji do zmieniających się preferencji użytkowników bez konieczności kosztownego re-treningu modelu.



Rysunek 4.4: Granice decyzyjne algorytmu KNN

Na rysunku 4.4 pokazano przykładowy wynik działania algorytmu. Przedstawione zostały granice decyzyjne, które określają do jakiej klasy zostaną zaklasyfikowane nowo dodane dane bez etykiet.

Do obliczenia k-najbliższych sąsiadów wykorzystywane są różne funkcje. Do intuicyjnych zalicza się między innymi odległość euklidesową. Dla odnalezienia odległości między p oraz q w n wymiarowej przestrzeni można obliczyć jako:

$$d_{Euklides}(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{4.13}$$

Do innych popularnych funkcji odległości należą odległość Manhattan oraz konsinusowa. Odległość Manhattan, znana również jako miejska lub ulicowa, jest definiowana jako:

$$d_{Manhattan}(p,q) = \sum_{i=1}^{n}|q_i - p_i| \tag{4.14}$$

Nazwa *odległość Manhattan* pochodzi od układu ulic na Manhattanie w Nowym Jorku, gdzie ulice są ułożone w regularną siatkę. W takim układzie, najkrótsza droga między dwoma punktami prowadzi przez siatkę ulic, a nie bezpośrednio w linii prostej, co odpowiada sumie różnic współrzędnych w poziomie i pionie.

Odległość kosinusowa jest szczególnie użyteczna przy porównywaniu wektorów w kontekście kąta pomiędzy nimi, co czyni ją popularnym wyborem w analizie tekstów i systemach rekomendacyjnych, gdzie ważniejsza jest orientacja niż wielkość wektorów. Odległość kosinusowa definiuje się jako:

$$d_{kosinusowa}(p,q) = 1 - \frac{\sum_{i=1}^{n} p_i \cdot q_i}{\sqrt{\sum_{i=1}^{n} p_i^2} \cdot \sqrt{\sum_{i=1}^{n} q_i^2}} \tag{4.15}$$

Każda z tych metryk ma swoje zastosowanie w zależności od charakterystyki danych i wymagań systemu rekomendacyjnego.

Algorytm KNN jest często stosowany w systemach rekomendacji treści ze względu na swoją prostotę i efektywność [92]. Jednak jego zastosowanie w dużych zbiorach danych napotyka na znaczące wyzwania, zwłaszcza w kontekście skalowalności i czasu obliczeń. W dużych systemach rekomendacyjnych, gdzie liczba użytkowników i elementów może osiągać wiele milionów, tradycyjne podejścia do KNN stają się nieefektywne z uwagi na konieczność obliczenia odległości nowego punktu względem każdego innego, co prowadzi do długich czasów odpowiedzi.

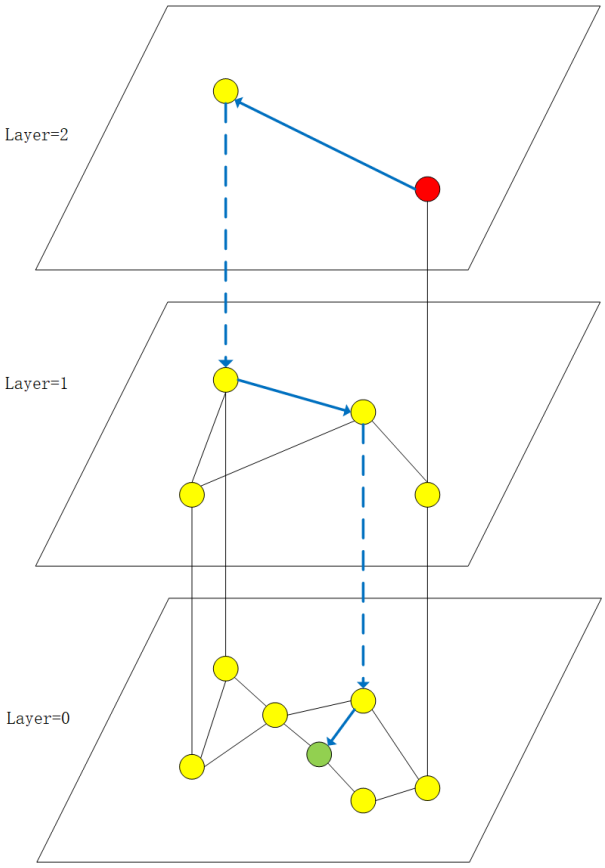
Ponadto, KNN jest szczególnie wrażliwy na problem *przekleństwa wymiarowości* (ang. *curse of dimensionality*), który pogarsza dokładność rekomendacji w przypadku obecności wielu cech gdzie każda cecha przekłada się na dodatkowy wymiar w przestrzeni. Dodatkowo nierównowaga w zbiorach danych, gdzie pewne klasy są nadreprezentowane, może również prowadzić do stronniczych wyników.

Aby radzić sobie z tymi wyzwaniami, stosuje się techniki takie jak redukcja wymiarowości za pomocą PCA, co pomaga zredukować czas obliczeń, zachowując przy tym istotne informacje. Metody równoważenia zbiorów danych, takie jak oversampling mniejszości lub undersampling większości, są wykorzystywane do zapobiegania stronniczości wyników. Także implementacja efektywnych struktur indeksowania, takich jak drzewa KD, może również znacząco przyspieszyć wyszukiwanie najbliższych sąsiadów w dużych zbiorach danych.

4.2.4 Approximate Nearest Neighbors - HNSW

Czasami czas uzyskania polecenia jest krytyczny i jednocześnie rozwiązanie problemu biznesowego może być przybliżonym wynik. Można wtedy skorzystać z modeli, które zwracają przybliżonych sąsiadów. W tym celu powstała cała rodzina algorytmów ANN (ang. *approximate nearest neighbors*). Popularnymi implementacjami ANN są HNSWlib [71], Annoy [34] i FAISS [33].

Algorytm Hierarchical Navigable Small World (HNSW) jest metodą opartą na grafach do wyszukiwania najbliższych sąsiadów w dużych zbiorach danych [71]. HNSW stosuje wielowarstwowe struktury grafów jak pokazano w przykładzie 4.5, gdzie każda kolejna warstwa ma coraz mniej węzłów, ale dłuższe krawędzie. Wyższe warstwy są używane do szybkiego przeszukiwania i lokalizacji obszarów, w których znajduje się zapytanie, podczas gdy niższe warstwy służą do dokładniejszego przeszukiwania lokalnych sąsiadów.



Rysunek 4.5: Wizualizacja przeszukiwania HNSW. Źródło: [65]

Przy dodawaniu nowego punktu do struktury, algorytm określa, na której warstwie powinien być on umieszczony, co zależy od losowo wygenerowanej wartości poziomu. Następnie, zaczynając od najwyższej warstwy, punkt jest dodawany do grafu, tworząc połączenia z najbliższymi sąsiadami w tej warstwie. Proces ten jest powtarzany dla każdej niższej warstwy aż do osiągnięcia poziomu przypisanego nowemu punktowi.

Wyszukiwanie rozpoczyna się od najwyższego poziomu grafu. Algorytm wybiera najbliższe sąsiednie węzły jako kandydatów, porusza się po krawędziach w dół do niższych warstw, stopniowo zmniejszając obszar poszukiwań. Ten proces jest kontynuowany aż do osiągnięcia najniższej warstwy, gdzie przeprowadza się dokładniejsze wyszukiwanie lokalne. Umożliwia to szybkie przeszukiwanie przestrzeni nawet w wysokowymiarowych danych, minimalizując potrzebę obliczania odległości do wszystkich punktów. Algorytm HNSW jest dostępny w różnych implementacjach, w tym w bibliotekach takich jak HNSWlib oraz jako część biblioteki FAISS od Meta.

4.3 Inne metody

Podczas pracy nad systemami rekomendacji treści pomocne mogą być również inne metody i algorytmy uczenia maszynowego [92]. W tej sekcji opisano ograniczone maszyny Boltzmanna (RBM), algorytm alokacji Dirichleta-Latenta (LDA) i analizę głównych składowych (PCA). Ograniczone maszyny Boltzmanna to nienadzorowane sieci neuronowe, które skutecznie wykrywają ukryte wzorce w danych. Algorytm alokacji Dirichleta-Latenta, jako generatywny model statystyczny, który identyfikuje tematy w tekstach, co pozwala na grupowanie dokumentów i rekomendowanie podobnych treści. Analiza składowych głównych, statystyczna technika redukcji wymiarowości, przekształca zestaw skorelowanych zmiennych w składowe główne, ułatwiając wizualizację i interpretację kluczowych wzorców danych.

4.3.1 Ograniczone maszyny Boltzmanna

Ograniczone maszyny Boltzmanna (ang. *restricted boltzmann machines (RBM)*) to rodzaj nienadzorowanej sieci neuronowej, która była wykorzystana w modelu łączonym BellKor podczas konkursu Netflix Prize [59]. Kluczową cechą RBM jest ich dwuwarstwowa struktura: jedna warstwa to neurony widoczne, które otrzymują dane wejściowe, a druga to neurony ukryte, które pomagają w odkrywaniu ukrytych wzorców w danych. Co ważne, w RBM nie ma połączeń między neuronami tej samej warstwy, co upraszcza model i proces uczenia [38].

W procesie uczenia RBM, celem jest dopasowanie modelu do danych w taki sposób, aby mógł on jak najlepiej odtworzyć lub wygenerować nowe dane, które są podobne do danych uczących. Uczenie odbywa się poprzez dostosowywanie wag między neuronami, co pozwala na modelowanie złożonych wzorców w danych. Jedną z unikalnych cech RBM jest to, że potrafią one uczyć się z danych bez konieczności nadzoru, co oznacza, że nie potrzebują etykietowanych danych do nauki.

Dodatkowo, biorąc pod uwagę istotny wpływ częstotliwości, warunkowane były one również w modelu. Pozwala to na uwzględnienie, jak często użytkownik ocenia filmy, co wpływa na jego percepcję. Zastosowane modyfikacje do wag widocznych-ukrytych, które były zależne od częstotliwości, pozwoliły na jeszcze lepsze dopasowanie modelu do danych.

Wszystkie te rozszerzenia modelu RBM były następnie wykorzystane w różnych wariantach włączonych do końcowego modelu łącznego, z różnymi liczbami iteracji i rozmiarami ukrytych jednostek, co miało na celu zwiększenie dokładności modelu. W finalnie połączonych modelach znalazły się zarówno RBM z warunkowaniem na podstawie daty i częstotliwości, jak i modele z mniejszą liczbą jednostek ukrytych, które były uruchomione przez większą liczbę iteracji.

4.3.2 LDA

LDA (ang. *Latent Dirichlet Allocation*) to generacyjny model statystyczny z rodziny NLP do modelowania tematów (ang. *topic modelling*) [56]. Model został zaproponowany przez Davida M. Blei, Andrew Y. Ng oraz Michaela I. Jordana [21]. Wykorzystywany jest w uczeniu maszynowym do odkrywania tematów w zbiorach dokumentów i ich późniejszemu grupowaniu. Dzięki identyfikacji pokrycia tematycznego dokumentów, LDA umożliwia rekomendowanie podobnych treści. W przeciwieństwie do k-means, które również pozwala przypisać *temat* do badanego dokumentu, LDA pozwala na przypisanie rozkładu wielu tematów w danym dokumencie. Głównym założeniem algorytmu jest to, że dokumenty są mieszanką wielu tematów w różnych proporcjach i tematy są mieszanką wielu słów.

Algorytm działa przez wyszukiwanie rozkładu słów współistniejących wśród dokumentów o podobnych tematach. W obliczeniach używane są dwa parametry α oznaczająca rzadkość tematów w dokumentach. Niższa wartość oznacza, że dokumenty będą miały mniej tematów. Oraz β, która kontroluje dystrybucję słów w tematach, gdzie niższe wartości sprzyjają większemu skupieniu na mniejszej liczbie słów. Poza systemami rekomendacji jest on używany powszechnie do analizy tekstów, wizualizacji danych czy klasyfikacji wiadomości.

- W pierwszym kroku wyznaczamy parametry α i β, następnie losowo przypisujemy słowa w dokumentach do tematów.
- Dla każdego słowa w każdym dokumencie obliczamy prawdopodobieństwo przypisania słowa do każdego tematu na podstawie obecnych przypisań.
- Finalnie przypisujemy słowo do tematu z największym prawdopodobieństwem.
- Proces powtarzany jest wielokrotnie aż do osiągnięcia zbieżności.

W przykładzie 4.3 przeprowadzono wyszukiwanie tematów i ich słów w dokumentach z wykorzystaniem modelu LDA z paczki Gensim [90].

Kod 4.3: LDA w Python 3

```
import gensim
import gensim.corpora as corpora
from gensim.models import LdaModel

documents = [
    "Lwy są wielkimi kotami żyjącymi w Afryce",
    "Psy są lojalnymi i przyjacielskimi zwierzętami domowymi",
    "Słonie to największe lądowe zwierzęta na świecie",
    "Tygrysy są znane z ich pasków i są drapieżnikami",
    "Koty są popularnymi zwierzętami domowymi ze względu na ich niezależność",
    "Samochody sportowe są szybkie i mają aerodynamiczny kształt",
    "SUVy są popularne ze względu na ich przestronność i wyższy prześwit",
    "Samochody elektryczne są przyjazne dla środowiska i mają niski poziom hałasu",
    "Silniki diesla są oszczędne, ale emitują więcej zanieczyszczeń",
    "Nowoczesne samochody mają wiele zaawansowanych technologii, takich jak systemy
        autonomiczne"
]

def preprocess(text):
    return gensim.utils.simple_preprocess(text, deacc=True)

processed_docs = [preprocess(doc) for doc in documents]
dictionary = corpora.Dictionary(processed_docs)
corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
lda_model = LdaModel(corpus=corpus, id2word=dictionary, num_topics=2, random_state=42,
     passes=10)

topics = lda_model.print_topics(num_words=5)
for topic in topics:
    print(topic)

new_doc = "Nowoczesne technologicznie samochody marki X oraz Y"
bow = dictionary.doc2bow(preprocess(new_doc))
topic_distribution = lda_model.get_document_topics(bow)
print(topic_distribution)
```

W podanym przykładzie 4.3 rozłożenie tematów wynosi odpowiednio 0.82 i 0.18. W pierwszym temacie dominujące słowa to *samochody, mają, są, nowoczesne, technologii*. W drugim temacie dominujące słowa to *są, ich, na, zwierzętami, domowymi*. Choć model poprawnie wykrył temat motoryzacyjny

jako dominujący to można go dalej usprawnić przez przetworzenie i usunięcie wyrazów nie wnoszących kontekstu.

4.3.3 PCA

Principal Component Analysis (PCA) jest techniką analizy statystycznej, która przekształca zestaw możliwie skorelowanych zmiennych w mniejszy zestaw zmiennych, zwanych głównymi składowymi [12]. Celem PCA jest redukcja wymiarowości danych przy minimalizacji utraty informacji [56]. Jest to osiągane poprzez projekcję danych na nową przestrzeń, gdzie pierwsza główna składowa wyjaśnia największą możliwą część wariancji danych, druga główna składowa wyjaśnia drugą największą część wariancji, i tak dalej.

Przed zastosowaniem PCA istotne jest przetworzenie danych. Na przykład, brakujące wartości w zbiorze danych mogą być sztucznie wstawione. Jednym z podejść jest sztuczne wstawianie przez średnią, gdzie brakujące wartości w danej kolumnie są zastępowane średnią tych wartości. Alternatywnie, można użyć sztuczne wstawianie medianą, która jest bardziej odporna na wpływ wartości odstających, ponieważ mediana jest mniej wrażliwa na skrajne wartości niż średnia. Istnieją również inne techniki sztucznego wstawiania, takie jak KNN-imputation (K-Nearest Neighbors), gdzie brakujące wartości są zastępowane średnią z najbliższych sąsiadów w przestrzeni cech.

Normalizacja danych jest krokiem w preprocessingu stosowanym, gdy cechy mają różne jednostki miary. Na przykład w zestawie danych, gdzie jedna cecha jest mierzona w kilogramach i inna w gramach, różnice w skali mogą prowadzić do nieadekwatnych wyników. Choć PCA może być stosowane bez standaryzacji, zaleca się ją w przypadku cech o różnych jednostkach miary, aby uzyskać bardziej obiektywne wyniki. PCA zakłada, że dane są skalowane, dlatego standaryzacja każdej cechy do średniej równej zero i jednostkowego odchylenia standardowego jest zalecana. Proces ten, zwany standaryzacją, przekształca dane w taki sposób, że każda cecha ma średnią równą zero i odchylenie standardowe równe jeden.

PCA działa poprzez obliczenie macierzy kowariancji danych oraz dekompozycję wartości własnych [23]. Niech X będzie macierzą danych z n próbkami i p cechami. Przed obliczeniem macierzy kowariancji, dane powinny być scentralizowane, czyli odjęta powinna być średnia każdej cechy:

$$X_{centered} = X - \bar{X} \tag{4.16}$$

Macierz kowariancji C jest dana przez:

$$C = \frac{1}{n-1} X_{centered}^{\top} X_{centered} \tag{4.17}$$

Wartości własne λ i wektory własne v macierzy C spełniają:

$$Cv = \lambda v \tag{4.18}$$

Wektory własne odpowiadające największym wartościom własnym tworzą główne składowe. Proces ten można również przeprowadzić za pomocą Singular Value Decomposition (SVD), co często jest bardziej efektywne obliczeniowo. Metoda SVD została opisana w innej części niniejszej pracy.

W systemie rekomendacji, niech R będzie macierzą interakcji użytkownik-przedmiot, gdzie wiersze reprezentują użytkowników i kolumny przedmioty. Stosując PCA do R, uzyskujemy zredukowaną reprezentację R', która uchwytuje najważniejsze wzorce w preferencjach użytkowników. Ta zredukowana reprezentacja może być użyta do tworzenia efektywnych i dokładnych rekomendacji.

Rozważmy system rekomendacji filmów z macierzą użytkownik-przedmiot R. Stosowanie PCA obejmuje następujące kroki:

- Centrowanie danych poprzez odjęcie średniej każdej cechy
- Obliczenie macierzy kowariancji danych scentralizowanych

- Przeprowadzenie dekompozycji wartości własnych macierzy kowariancji
- Wybór top k wektorów własnych do utworzenia głównych składowych
- Rzutowanie oryginalnych danych na główne składowe w celu uzyskania zredukowanej macierzy R'

Wariancja wyjaśniona przez każdą główną składową jest kluczowa dla interpretacji wyników PCA. Wariancję tę można obliczyć jako stosunek wartości własnej danej składowej do sumy wszystkich wartości własnych:

$$explained = \frac{\lambda_i}{\sum_{i=1}^{p} \lambda_i} \tag{4.19}$$

Kiedy decydujemy, ile głównych składowych zachować, używamy wykresu osypiska (ang. *scree plot*), który pokazuje wartości własne w kolejności malejącej. Na wykresie osypiska obserwujemy, gdzie następuje *załamanie*, czyli punkt, w którym wartości własne przestają gwałtownie maleć i stają się bardziej płaskie. Liczba składowych przed tym załamaniem jest często wybierana jako optymalna liczba głównych składowych.

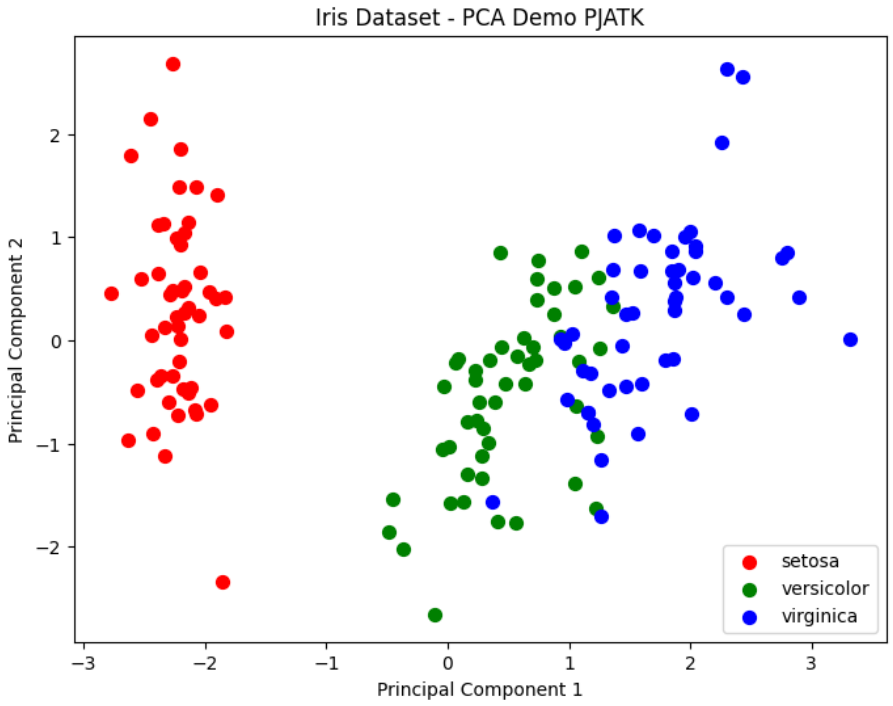
W przykładzie 4.4 pokazano przykładowe wykonanie PCA na zestawie danych *iris* [39].

Kod 4.4: PCA w Python 3

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

iris = load_iris()
X = iris.data
y = iris.target

from sklearn.preprocessing import StandardScaler
X = StandardScaler().fit_transform(X)

pca = PCA(n_components=2)
principal_components = pca.fit_transform(X)

principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

final_df = pd.concat([principal_df, pd.DataFrame(y, columns=['target'])], axis=1)

plt.figure(figsize=(8, 6))
targets = [0, 1, 2]
colors = ['r', 'g', 'b']
for target, color in zip(targets, colors):
   indices_to_keep = final_df['target'] == target
   plt.scatter(final_df.loc[indices_to_keep, 'PC1'],
       final_df.loc[indices_to_keep, 'PC2'],
       c=color,
       s=50)
plt.legend(iris.target_names)
plt.xlabel('Principal Component 1')
```

```
plt.ylabel('Principal Component 2')
plt.title('Iris Dataset - PCA Demo PJATK')
plt.show()

print(pca.explained_variance_ratio_)
```

Wynik kodu 4.4 dla zestawu danych Iris w formie dwóch wymiarów PCA został zwizualizowany na Rysunku 4.6, wariancja wyjaśniona przez PCA wynosi $[0.72962445, 0.22850762]$. Oznacza to, że pierwsza główna składowa wyjaśnia około 73% całkowitej wariancji, a druga około 23%. Suma tych dwóch wartości wskazuje, że dwie główne składowe wyjaśniają około 96% całkowitej wariancji danych, co sugeruje, że większość istotnej informacji w zbiorze danych jest uchwycona przez te dwie składowe. Na Rysunku 4.6 możemy zaobserwować, że różne gatunki kwiatów tworzą wyraźne skupiska w zredukowanej przestrzeni, co wskazuje na skuteczność algorytmu PCA.



Rysunek 4.6: Wynik PCA dla zestawu danych Iris

PCA ma swoje ograniczenia, które należy brać pod uwagę:

- PCA zakłada, że zmiany w danych można uchwycić za pomocą liniowych kombinacji zmiennych, co może być niewystarczające dla bardziej złożonych struktur danych.
- Wartości własne są wrażliwe na skalowanie cech, dlatego dane muszą być odpowiednio przeskalowane przed zastosowaniem PCA.
- Główne składowe mogą być trudne do interpretacji, szczególnie jeśli zmienne oryginalne mają różne jednostki.

4.4 Biblioteki i paczki w środowisku Python

W niniejszym rozdziale przedstawiono wybrane biblioteki dostępne w języku Python 3, które są przydatne w zadaniach związanych z uczeniem maszynowym i przetwarzaniem języka naturalnego. Omówiono bibliotekę Scikit Learn [96], która dostarcza zestaw narzędzi do budowy modeli uczenia maszynowego, takich jak modele regresji, klasyfikacji czy grupowania. Zaprezentowano również bibliotekę Surprise [101], służącą do tworzenia i analizy systemów rekomendacji, umożliwiającą przetwarzanie danych dotyczących ocen użytkowników. Wreszcie, przedstawiono bibliotekę NLTK [19], która oferuje zestaw narzędzi do przetwarzania i analizy języka naturalnego, w tym tokenizatory, lematyzatory oraz rozbudowane korpusy językowe.

4.4.1 Scikit Learn

Biblioteka Scikit Learn [96] znana również jako sklearn to otwarta biblioteka do Pythona, zaprojektowana do rozwiązywania zadań uczenia maszynowego i analizy danych. Zawiera proste i efektywne narzędzia i modele gotowe do użycia. Scikit Learn opiera się na w głównej mierze na dwóch innych - NumPy oraz SciPy. Biblioteka została stworzona przez Davida Cournapeau jako projekt Google Summer of Code w 2007 roku [82]. Następnie Matthieu Brucher pracował nad biblioteką w swojej pracy dyplomowej.

Scikit-learn oferuje różne modele i algorytmy [20]. Między innymi takie jak `DecisionTreeClassifier` i `DecisionTreeRegressor` dla zadań klasyfikacji i regresji, umożliwiające modelowanie zależności nieliniowych. Modele `LinearRegression` i `LogisticRegression` są wykorzystywane odpowiednio do przewidywania wartości ciągłych oraz klasyfikacji binarnej. `KMeans` jest algorytmem stosowanym do grupowania danych, wykorzystującym metodę iteracyjną do minimalizacji sumy kwadratów odległości od centroidów. Analiza głównych składowych (`PCA`) służy do redukcji wymiarów danych przez identyfikację i zachowanie kierunków, które maksymalizują wariancję. W dziedzinie optymalizacji, algorytm gradientu prostego jest realizowany przez klasy `SGDRegressor` i `SGDClassifier`. Biblioteka oferuje też metody regularyzacji takie jak L1 (lasso) i L2 (ridge), które pomagają w zapobieganiu przeuczeniu modelu.

Do skalowania danych, Scikit-learn dostarcza narzędzia takie jak `StandardScaler` i `MinMaxScaler`, które normalizują dane poprzez usunięcie średniej i skalowanie do jednostkowej wariancji, lub skalowanie atrybutów do określonego zakresu. Metryki takie jak precyzja, pełność i wynik F1 są obliczane za pomocą funkcji `precision_score`, `recall_score`, i `f1_score`. Funkcja `cross_val_score` pozwala na ocenę modelu za pomocą walidacji krzyżowej.

W przykładzie 4.5 pokazano użycie biblioteki Scikit Learn.

Kod 4.5: Użycie modeli z biblioteki Scikit Learn w języku Python 3

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

bc = datasets.load_breast_cancer()
X = bc.data
y = bc.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state
    =1)
```

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

models = [
    ('Logistic Regression', LogisticRegression(random_state=1)),
    ('SVC', SVC(kernel='rbf', C=1.0, random_state=1)),
    ('Random Forest', RandomForestClassifier(n_estimators=100, random_state=1))
]

for name, model in models:
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"Dokładność modelu {name} wynosi {accuracy}")
```

W ten sposób przedstawiono, jak można wykorzystać modele regresji logistycznej, wektorów nośnych i losowego lasu decyzyjnego do przewidywania wyników badań dotyczących raka piersi z wykorzystaniem zbioru danych Wisconsin [107]. Wynik działania zwraca odpowiednio dla modelu Logistic Regression dokładność 0.9736842105263158, dla modelu SVC będącego implementacją modeli nośnych dokładność 0.9736842105263158 i dla modelu Random Forest dokładność 0.956140350877193. Zaimplementowany przykład pokazuje pełny proces: od podzielenia danych na zestawy treningowe i testowe, poprzez trenowanie modelu, aż po prognozowanie wyników i ocenę modelu.

4.4.2 Surprise

Surprise to biblioteka Pythona służąca do budowy i analizy systemów rekomendacji [101]. Pozwala na łatwe przetwarzanie danych dotyczących ocen użytkowników i budowanie na ich podstawie zaleceń. Biblioteka umożliwia ładowanie zestawów danych, przetwarzanie ich, dzielenie na zestawy do trenowania i testowania oraz stosowanie różnych algorytmów predykcji ocen. Biblioteka powstała jako część pracy Nicolasa Hug w 2020 roku w Columbia University [49].

Wśród wybranych elementów biblioteki Surprise można znaleźć `Dataset`, klasę, która stanowi centralny punkt zarządzania danymi w bibliotece Surprise. Umożliwia wczytywanie danych z różnorodnych źródeł, takich jak pliki w formacie CSV, dataframe z biblioteki Pandas, czy innych typów źródeł danych. Klasa ta zapewnia metody takie jak `load_from_file()`, `load_from_df()`, które pozwalają na łatwe wczytanie danych i stosowanie funkcji `build_full_trainset()` do konwersji danych na wewnętrzny format, który jest optymalizowany pod kątem przetwarzania w algorytmach Surprise.

`Reader` jest pomocniczą klasa konfiguracyjna, która informuje `Dataset` o formacie i zakresie danych wejściowych. Za pomocą parametrów takich jak `line_format` (definiujący kolejność i typy danych w każdej linii danych) i `rating_scale` (określający minimalną i maksymalną możliwą ocenę), `Reader` dostosowuje wczytywanie danych do potrzeb użytkownika, co jest kluczowe dla prawidłowego przetwarzania i analizy.

Funkcja `train_test_split` pozwala na podzielenie danych na zestawy treningowe i testowe w dowolnych proporcjach, zwykle z domyślnym podziałem 75% na trening i 25% na testy. `cross_validation`, z kolei, jest metodą walidacji, która iteracyjnie dzieli cały zestaw danych na k podzbiorów, pozwalając na przetestowanie modelu na różnych kombinacjach tych podzbiorów.

Poza tym biblioteka Surprise posiada wbudowane modele, między innymi takie jak `SVD (Singular Value Decomposition)`, `KNNBasic`, i `NMF (Non-negative Matrix Factorization)`. `SVD`

jest techniką faktoryzacji macierzy wykorzystywaną do dekompozycji macierzy interakcji użytkowników z produktami na trzy mniejsze macierze, co pozwala na wydobycie latentnych czynników wpływających na oceny. `KNNBasic` to implementacja algorytmu k-najbliższych sąsiadów (k-NN) wykorzystująca różne miary podobieństwa, takie jak odległość euklidesowa, cosinusowa, czy współczynnik korelacji Pearsona. `NMF` to metoda faktoryzacji macierzy, która dzieli macierz ocen na macierze z nieujemnymi wartościami, co jest szczególnie przydatne, gdy dane są naturalnie ograniczone do nieujemności, jak w przypadku ocen. Dodatkowo, `BaselineOnly` to algorytm, który wykorzystuje prostą metodę obliczania bazowych przewidywań ocen, stosując średnie globalne oraz średnie oceny użytkowników i produktów.

Kod 4.6: Wykorzystanie modelu SVD z Surprise w Python 3

```
from surprise import SVD
from surprise import Dataset
from surprise.model_selection import cross_validate

data = Dataset.load_builtin('ml-100k')

algo = SVD()

cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

Za pomocą modelu SVD w przykładzie 4.6, dane są ładowane z wbudowanego zbioru `ml-100k` i przeprowadzana jest walidacja krzyżowa, której celem jest ocena dokładności modelu przy użyciu miar RMSE oraz MAE. Wyniki są wyświetlane dla każdego z pięciu podziałów zestawu danych, co pozwala na ocenę efektywności modelu w przewidywaniu ocen.

4.4.3 NLTK

NLTK (ang. *natural language toolkit*) to biblioteka w Pythonie 3 napisana przez Stevena Birda, Edwarda Lopera i Ewana Kleina w 2001 roku służąca do pracy z językiem naturalnym [19]. Zawiera popularne narzędzia w dziedzinie przetwarzania naturalnego NLP służące przykładowo do tokenizacji, tagowania części mowy, analizy składniowej czy wydobywania informacji. Większość modeli i funkcji NLTK opiera się na statystykach tekstowych takich jak częstość występowania słów i fraz, rozkłady prawdopodobieństwa i współwystępowania.

Do kluczowych komponentów biblioteki można zaliczyć tokenizery, czyli narzędzia, które dzielą tekst na mniejsze jednostki zwane tokenami. Tokeny mogą być pojedynczymi słowami, frazami lub symbolami. NLTK udostępnia metodę `word_tokenize` oraz tokenizator Punkt, który jest dostosowany do wielu zachodnich języków i bazuje na zasadach ortograficznych tych języków. Jest on także skuteczny w wyłapywaniu skomplikowanych przypadków takich jak skróty i skrótowce. `sent_tokenize` działa podobnie lecz dzieli tekst na zdania. Tokenizer `PunktSentenceTokenize` to klasyfikator uczenia maszynowego, który uczy się na konkretnym korpusie tekstowym.

Inny istotny komponent to *Lematyzatory*. Przekształcają one słowa do ich leksykalnej formy podstawowej. W przeciwieństwie do stemingu, który agresywnie obcina końcówki słów, lematyzacja wykorzystuje kontekst do transformacji słów. W NLTK można skorzystać z lematyzatora `WordNetLemmatizer`, który używa bazy danych WordNet do wyszukiwania lematów. Jest szczególnie użyteczny dla języka angielskiego, gdzie baza WordNet dostarcza obszerne informacje o relacjach między słowami. Na przykład, słowo `better` zostanie przekształcone do `good`.

Trzeci komponent NLTK w kontekście NLP to korpusy językowe. Służą one jako materiał źródłowy do trenowania i testowania algorytmów NLP. Dostępne w NLTK są przykładowo korpusy językowe takie jak `reuters` czy `gutenberg`, które zawierają duże zbiory tekstów literackich lub artykułów prasowych. Służą

one do analiz statystycznych, uczenia modeli językowych oraz eksperymentów z analizą semantyczną. Inne korpusy takie jak brown dostarczają teksty z tagami części mowy i innymi metadanymi. Są one używane przy trenowaniu taggerów czy parserów składniowych.

Kod 4.7: Tokenizacja słów za pomocą NLTK w Python 3

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

text = "Good morning PJATK!"

tokens = word_tokenize(text)
print(tokens)
```

W przykładzie 4.7 ukazano podstawowe użycie funkcji word_tokenize do podziału zdania na tokeny. Funkcja ta jest częścią modułu nltk.tokenize, który został zaimportowany na początku skryptu. Pierwsza linia kodu pobiera niezbędny zasób punkt, co jest wymagane do poprawnego działania tokenizatora. Następnie zadeklarowano zmienną text zawierającą przykładowe zdanie do analizy. Funkcja word_tokenize dzieli to zdanie na listę tokenów, którą następnie wyświetla na ekranie za pomocą funkcji print.

5. Implementacja prototypu mikrousługi

Na potrzeby niniejszej pracy dyplomowej utworzono prototyp mikrousługi systemu rekomendacji treści. System został przygotowany w języku programowania Python 3.10 [87] z wykorzystaniem frameworka do tworzenia serwerów HTTP Flask w wersji 2.3.2 [79]. Relacyjna baza danych, która przechowuje dane o użytkownikach, przedmiotach rekomendacji i ocenach użytkowników, jest napędzana za pomocą silnika bazodanowego PostgreSQL 15.2 [102]. Połączenie z bazą danych jest zapewnione przez bibliotekę Psycopg 2.9 [27]. Komunikacja z mikrousługą odbywa się za pomocą protokołu HTTP w architekturze REST, a klient HTTP wykorzystywany do testowania to Insomnia [57]. Kod źródłowy przechowywany jest w repozytorium Git na platformie GitHub [41].

W dalszej części rozdziału omówiono komponenty i moduły systemu rekomendacji treści. Na początku przedstawiono schemat bazy danych, w tym relacje między tabelami i typy danych. Omówiono sposób modelowania danych użytkownika, obiektów rekomendacji i ocen i sposób ich przechowywania i zarządzania nimi. Kolejna sekcja koncentruje się na warstwie dostępu do danych, opisując wykorzystanie biblioteki Psycopg, modeli danych i zapytań SQL, które umożliwiają wydajne i bezpieczne operacje na bazie danych.

Dalsza część rozdziału poświęcona jest warstwie kontrolerów, które przekształcają żądania HTTP na konkretne działania w aplikacji. Opisano tutaj użyte wzorce projektowe i architektoniczne, takie jak wzorzec model-widok-kontroler (ang. *Model-View-Controller, MVC*) oraz wstrzykiwanie zależności (ang. *dependency injection*), które ułatwiły rozwój systemu oraz przyczyniły się do jego modularności.

Zajęto się również dynamicznym ładowaniem modułów zawierających algorytmy uczenia maszynowego. Mechanizm ten pozwala na ich dynamiczne dodawanie i modyfikowanie bez przerywania pracy systemu. Omówiono zastosowane rozwiązania techniczne, które umożliwiają integrację nowych algorytmów i ich konfigurację w trakcie działania systemu.

Przedostatnia sekcja poświęcona jest przeprowadzonym testom. Opisano różne rodzaje testów, które zostały wdrożone, w tym testy jednostkowe, integracyjne i wydajnościowe. Omówiono narzędzia wykorzystane do testowania i strategie zapewniające wysoką jakość kodu.

Zakończenie rozdziału dotyczy budowy obrazu Dockera i jego dystrybucji na platformie DockerHub [31]. Przedstawiono proces tworzenia konteneryzowanego środowiska aplikacji, omówiono zalety wykorzystania Dockera oraz opisano kroki niezbędne do wdrożenia obrazu na platformie DockerHub. Dzięki temu zapewniono możliwość skalowania i wdrażania aplikacji na różnych środowiskach.

5.1 Mikrousługa

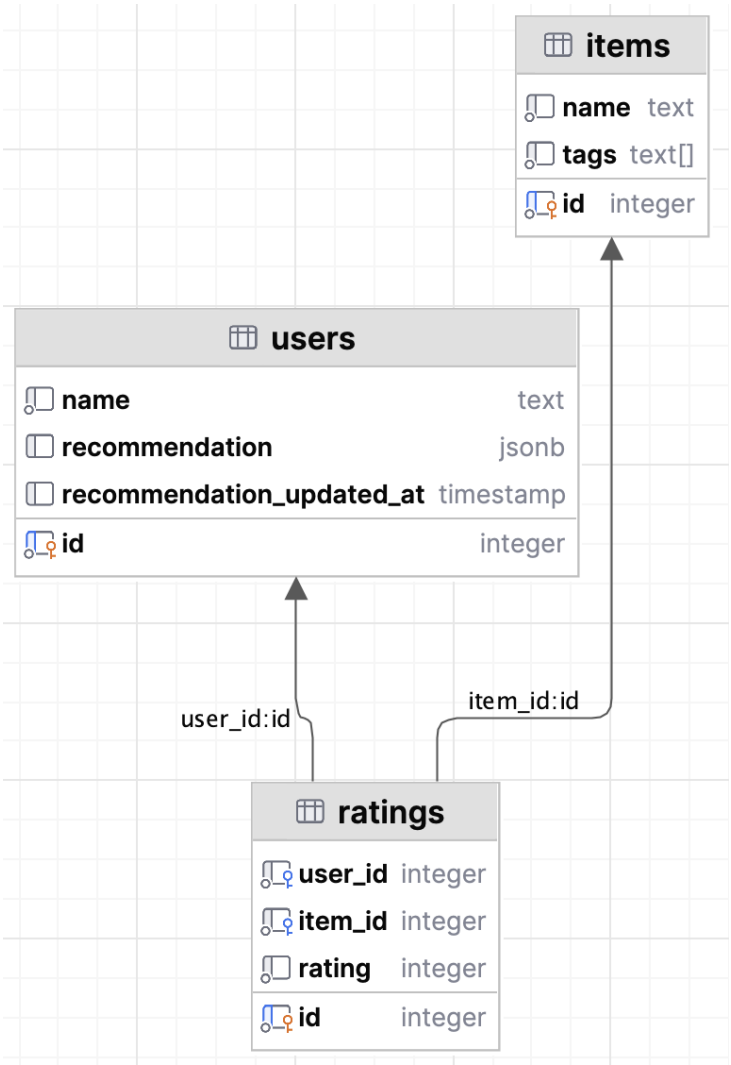
Na potrzeby systemu, część odpowiedzialną za obliczenia rekomendacji wykonano w architekturze mikusługi. Architektura mikrousług jest szczególnie odpowiednia dla silnika rekomendacji, systemu z natury złożonego i często wymagającego szybkiego, niezależnego skalowania w porównaniu do innych części aplikacji biznesowej. W przeciwieństwie do architektury monolitycznej, w której wszystkie komponenty są ze sobą ściśle powiązane i skalowane jako pojedyncza jednostka, mikrousługi pozwalają na rozwijanie, wdrażanie i skalowanie silnika rekomendacji niezależnie od reszty aplikacji [45]. Jest to korzystne, ponieważ silniki rekomendacji często muszą przetwarzać duże ilości danych i wymagają częstych aktualizacji w oparciu o interakcje użytkowników i informacje zwrotne, które mogą być obsługiwane bardziej efektywnie, gdy są oddzielone od reszty systemu.

Co więcej dzięki zastosowaniu izolacji poszczególnych komponentów, ewentualna awaria silnika rekomendacji nie wpływa negatywnie na funkcjonalność pozostałych elementów aplikacji. W przeciwieństwie do podejścia monolitycznego, gdzie błąd w jednym module może skutkować całkowitym przestojem systemu, architektura mikrousług minimalizuje ryzyko globalnych awarii. Co więcej, takie rozwiązanie umożliwia wykorzystanie zróżnicowanych technologii, zoptymalizowanych pod kątem spe-

cyficznych zadań realizowanych przez silnik rekomendacji, takich jak przetwarzanie danych w czasie rzeczywistym, analiza danych czy generowanie spersonalizowanych treści. Elastyczność w doborze języków programowania oraz systemów przechowywania danych, dostosowanych do konkretnych wymagań silnika rekomendacji, przekłada się na wyższą jakość i trafność generowanych rekomendacji. W rezultacie, zastosowanie architektury mikrousług w tym kontekście przyczynia się do poprawy ogólnych wrażeń użytkowników i zwiększenia ich satysfakcji z korzystania z systemu.

5.2 Baza danych

W aplikacji zastosowano relacyjną bazę danych PostgreSQL [102]. Schemat przedstawiony na rysunku 5.1 składa się z tabel `users`, `ratings` i `items`.



Rysunek 5.1: Schemat bazy danych

Tabela `users` o strukturze jak w 5.1 przechowuje informacje o użytkownikach końcowych, dla których generowane są rekomendacje. Składa się ona z kolumn: `id` - identyfikator użytkownika w systemie, `name` - nazwa użytkownika lub jego identyfikator zewnętrzny na innej platformie (np. w serwisie rekomendującym książki, identyfikator może pochodzić z tej platformy), `recommendation` typu jsonb, zawierająca zapis wyników poprzednich obliczeń rekomendacji, co umożliwia aplikacji ciągłe

dostarczanie rekomendacji, nawet podczas trenowania nowego zestawu danych, oraz `recommendation_updated_at` typu timestamp, z datą ostatniej aktualizacji rekomendacji.

Tabela 5.1: Struktura tabeli users

Kolumna	Typ	Opis
id	integer	Identyfikator użytkownika w systemie
name	text	Nazwa lub identyfikator zewnętrzny użytkownika w obcym serwisie
recommendation	jsonb	Zapisany wynik poprzednich obliczeń rekomendacji
recommendation_update_at	timestamp	Data ostatniej aktualizacji rekomendacji

Tabela bazodanowa `ratings` o strukturze jak pokazano w tabeli 5.2 służy do przechowywania ocen udzielonych przez użytkowników poszczególnym przedmiotom. Składa się z kolumn: `user_id` typu integer, identyfikującej użytkownika, który udzielił oceny, `item_id` typu integer, identyfikującej oceniany przedmiot, oraz `rating` typu integer, przechowującej ocenę produktu. Skala ocen jest ustalana indywidualnie zgodnie z potrzebami aplikacji.

Tabela 5.2: Struktura tabeli ratings

Kolumna	Typ	Opis
user_id	integer	Identyfikator użytkownika, który udzielił oceny
item_id	integer	Identyfikator przedmiotu ocenianego
rating	integer	Ocena przedmiotu (skala nie jest narzucona przez system)

Struktura tabeli w bazie danych `items` opisana także w tabeli 5.3 zawiera informacje o przedmiotach, które są oceniane, takich jak filmy, książki czy utwory muzyczne. Zawiera kolumny: `id` typu integer z identyfikatorem przedmiotu, `name` typu text z nazwą przedmiotu lub jego identyfikatorem zewnętrznym, oraz `tags` typu tablicy text, która przechowuje etykiety przyporządkowane do przedmiotu. Etykiety są wykorzystywane do dodatkowego przetwarzania w celu określenia stopnia podobieństwa między przedmiotami na podstawie wspólnych etykiet.

Tabela 5.3: Struktura tabeli items

Kolumna	Typ	Opis
id	integer	Identyfikator przedmiotu
name	text	Nazwa lub identyfikator zewnętrzny przedmiotu
tags	array of text	Tablica etykiet przyporządkowanych do przedmiotu

Relacja jeden-do-wielu pomiędzy tabelami `users` i `ratings` oznacza, że pojedynczy użytkownik (reprezentowany przez rekord w tabeli `users`) może udzielić wielu ocen różnym przedmiotom. Każda ocena (reprezentowana przez rekord w tabeli `ratings`) jest przypisana do jednego użytkownika. W praktyce oznacza to, że w tabeli `ratings` jest klucz obcy `user_id`, który odnosi się do klucza głównego `id` w tabeli `users`. Klucz obcy zapewnia integralność referencyjną danych, umożliwiając śledzenie, który użytkownik udzielił danej oceny.

Podobnie, każdy przedmiot (reprezentowany przez rekord w tabeli `items`) może być oceniany wielokrotnie przez różnych użytkowników. Każda ocena w tabeli `ratings` jest przypisana do jednego

przedmiotu. Tutaj również klucz obcy `item_id` w tabeli `ratings` odnosi się do klucza głównego `id` w tabeli `items`, co pozwala identyfikować przedmiot, który został oceniony.

Do aplikowania zmian w bazie danych przygotowano wyspecjalizowaną usługę nazwaną `MigrationManager`. Funkcja `run_migration` odpowiada za aplikowanie skryptów migracji bazy danych. Jej celem jest automatyzacja procesu aktualizacji schematu bazy danych poprzez sekwencyjne wykonywanie skryptów SQL zawartych w plikach znajdujących się w określonym katalogu.

Wewnątrz funkcji, następuje iteracja po wszystkich plikach znajdujących się w określonym katalogu `directory`. Pliki są sortowane alfabetycznie, aby zapewnić właściwą kolejność wykonywania skryptów migracji. Dla każdego pliku o rozszerzeniu ".sql", wykonywane są następujące kroki:

Warto zauważyć, że funkcja korzysta z biblioteki `dotenv` do ładowania zmiennych środowiskowych z pliku `.env`. Zmienna środowiskowa `VERBOSE` jest wykorzystywana do kontrolowania, czy informacje o wykonywanych skryptach migracji mają być wypisywane na standardowe wyjście.

5.3 Warstwa dostępu do danych

Do komunikacji z bazą danych opisaną w innej części niniejszej pracy wykorzystywana jest biblioteka Psycopg [27]. Wykorzystuje ona pod spodem bibliotekę `libpq`, która jest oficjalnym klientem Postgresa w języku Python 3 [84]. Psycopg charakteryzuje się modułami napisanymi w języku C, które służą przyspieszeniu w komunikacji. Wspiera Pythona od wersji Python 3.7 i PostgreSQL od wersji 7.4.

Z jej wykorzystaniem utworzono paczkę models, które zawiera reprezentacje encji pod postacią klas Python 3. W kodzie 5.1 pokazano przykładowy model reprezentujący tabelę `items`.

Kod 5.1: Model Items w języku Python 3

```
class Item:
    id: int
    name: str
    tags: list[str]

    def __init__(self, id: int, name: str, tags: list[str]):
        self.id = id
        self.name = name
        self.tags = tags

    def __repr__(self):
        return f"Item(id={self.id}, name={self.name}, tags={self.tags})"
```

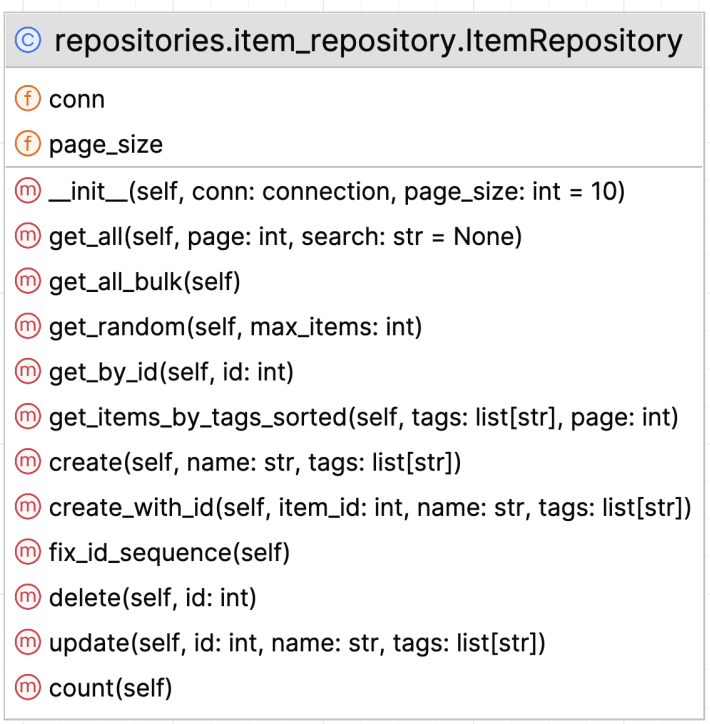
Do wykonywania kwerend SQL utworzono warstwę repozytorium, która zapewnia abstrakcję nad operacjami bazodanowymi i umożliwia łatwiejszą zmianę źródeł danych bez wpływu na logikę biznesową aplikacji. Klasy w tej warstwie są odpowiedzialne za bezpośrednią komunikację z bazą danych, wykonując kwerendy, aktualizacje oraz operacje usuwania. Przykładowy fragment implementacji klasy repozytorium dla modelu Item został pokazany w kodzie 5.2.

Kod 5.2: Fragment repozytorium `ItemRepository`

```
class ItemRepository:
    def __init__(self, conn: connection, page_size: int = 10):
        self.conn = conn
        self.page_size = page_size

    def get_all(self, page: int, search: str = None) -> list[Item]:
```

```
with self.conn.cursor() as cursor:
    if search is None:
        cursor.execute("SELECT * FROM items LIMIT %s OFFSET %s", (self.page_size
            , page * self.page_size,))
    else:
        cursor.execute("SELECT * FROM items WHERE name ILIKE %s LIMIT %s OFFSET
            %s", (f"%{search}%", self.page_size, page * self.page_size,))
    return [Item(*row) for row in cursor.fetchall()]
```

Na rysunku 5.2 przedstawiono diagram reprezentujący klasę ItemRepository wraz ze wszystkimi metodami i polami.

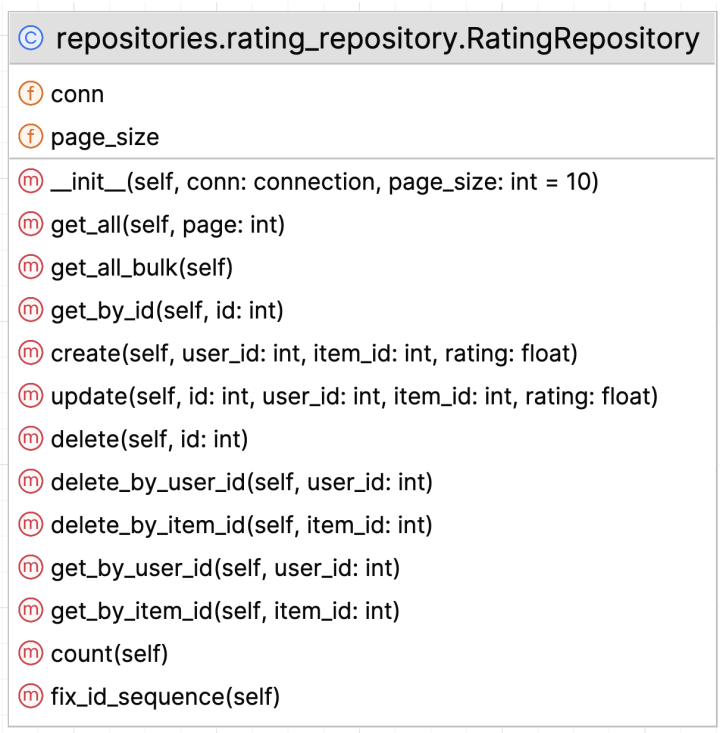


Rysunek 5.2: Diagram klasy `ItemRepository`

- Metoda `get_all` służy do pobrania wszystkich przedmiotów na danej stronie. Opcjonalnie można przekazać wyszukiwaną nazwę lub jego fragment do wyszukania. Stronicowanie wykonano za pomocą operatorów LIMIT-OFFSET.
- Metoda `get_all_bulk` pobiera i zwraca wszystkie przedmioty z bazy danych bez stronicowania.
- Metoda `get_random` losowo wybiera określoną liczbę przedmiotów z bazy.
- Metoda `get_by_id` zwraca przedmiot o podanym identyfikatorze; jeśli przedmiot nie istnieje, zwraca None.
- Metoda `get_items_by_tags_sorted` zwraca listę przedmiotów pasujących do podanych tagów, posortowanych najpierw według liczby pasujących tagów i potem według liczby wysokich ocen.
- Metoda `create` dodaje nowy przedmiot do bazy danych i zwraca go.
- Metoda `create_with_id` działa podobnie, ale pozwala określić identyfikator nowego przedmiotu.

- Metoda `fix_id_sequence` aktualizuje sekwencję identyfikatorów przedmiotów, aby odpowiadała maksymalnemu identyfikatorowi w bazie danych.
- Metoda `delete` usuwa przedmiot o podanym identyfikatorze z bazy danych.
- Metoda `update` aktualizuje nazwę i tagi przedmiotu o określonym identyfikatorze i zwraca zaktualizowany przedmiot.
- Metoda `count` zlicza i zwraca liczbę przedmiotów w bazie danych.

Diagram z rysunku 5.3 przedstawia klasę `RatingRepository` wraz ze wszystkimi metodami i polami.

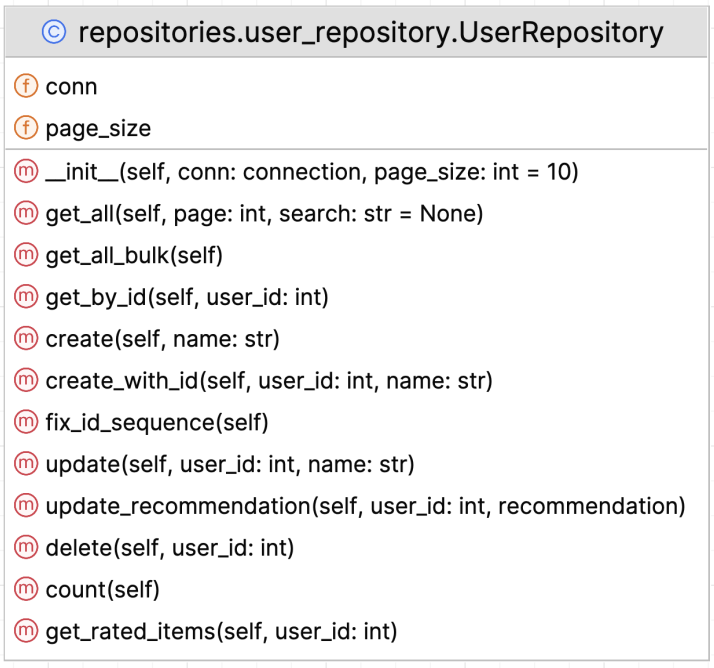


Rysunek 5.3: Diagram klasy `RatingRepository`

- Metoda `get_all` służy do pobrania stronicowanej listy ocen z bazy danych. Wykorzystuje LIMIT i OFFSET do paginacji wyników.
- Metoda `get_all_bulk` zwraca wszystkie oceny z bazy danych bez stronicowania.
- Metoda `get_by_id` zwraca ocenę o podanym identyfikatorze; jeśli ocena nie istnieje, zwraca None.
- Metoda `create` dodaje nową ocenę do bazy danych i zwraca utworzoną ocenę.
- Metoda `update` aktualizuje istniejącą ocenę, zmieniając jej składowe i zwraca zaktualizowaną ocenę.
- Metoda `delete` usuwa ocenę o podanym identyfikatorze z bazy danych.
- Metoda `delete_by_user_id` usuwa wszystkie oceny przypisane do określonego użytkownika.
- Metoda `delete_by_item_id` usuwa wszystkie oceny przypisane do określonego przedmiotu.

- Metoda `get_by_user_id` zwraca wszystkie oceny wystawione przez użytkownika o podanym identyfikatorze.
- Metoda `get_by_item_id` zwraca wszystkie oceny przypisane do przedmiotu o podanym identyfikatorze.
- Metoda `count` zlicza i zwraca liczbę wszystkich ocen w bazie danych.
- Metoda `fix_id_sequence` aktualizuje sekwencję identyfikatorów ocen, aby odpowiadała maksymalnemu identyfikatorowi w bazie danych.

Finalnie, na rysunku 5.4 pokazano diagram klasy `UserRepository`.



Rysunek 5.4: Diagram klasy `UserRepository`

- Metoda `get_all` pozwala na stronicowane pobieranie użytkowników z bazy danych. Można opcjonalnie przekazać ciąg wyszukiwania, aby filtrować użytkowników po nazwie.
- Metoda `get_all_bulk` zwraca listę wszystkich użytkowników z bazy danych bez paginacji.
- Metoda `get_by_id` zwraca użytkownika o podanym identyfikatorze; jeśli użytkownik nie istnieje, zwraca None.
- Metoda `create` tworzy nowego użytkownika z podaną nazwą i zwraca go.
- Metoda `create_with_id` pozwala na stworzenie użytkownika z określonym identyfikatorem oraz nazwą i zwraca go.
- Metoda `fix_id_sequence` aktualizuje sekwencję identyfikatorów użytkowników, aby odpowiadała maksymalnemu identyfikatorowi w bazie danych.
- Metoda `update` aktualizuje nazwę istniejącego użytkownika o podanym identyfikatorze i zwraca zaktualizowanego użytkownika.

- Metoda `update_recommendation` aktualizuje rekomendacje dla użytkownika, konwertując dane na format JSON, jeśli jest to konieczne, i zapisując czas aktualizacji.
- Metoda `delete` usuwa użytkownika o podanym identyfikatorze z bazy danych.
- Metoda `count` zlicza i zwraca liczbę użytkowników w bazie danych.
- Metoda `get_rated_items` zwraca listę przedmiotów ocenionych przez użytkownika, łącząc informacje o przedmiotach z ocenami.

5.4 Kontrolery

Aplikacja została przygotowana w architekturze model-widok-kontroler (ang. *Model-View-Controller, MVC)*. Składa się ona z trzech warstw, modelu z dostępem do bazy danym, widoku z logiką zwracającą dane użytkownikowi w zadanym formacie oraz kontrolerów [28]. Kontrolery odbierają żądania od użytkowników przez HTTP i na ich podstawie podejmują decyzje dotyczące dalszego przetwarzania. W zależności od danych wejściowych i parametrów żądania, kontroler może wywołać odpowiednie metody modelu w celu pobrania danych lub zmiany stanu aplikacji.

Innym zadaniem kontrolerów jest przetwarzanie danych otrzymanych z modelu. Kontroler analizuje te dane i podejmuje decyzje dotyczące sposobu ich dalszej prezentacji [62]. Określa, które widoki są potrzebne do wyświetlenia odpowiedzi użytkownikowi oraz jakie dodatkowe przetwarzanie danych jest konieczne przed ich prezentacją. Ta decyzyjna rola kontrolerów umożliwia odpowiednie dostosowanie odpowiedzi aplikacji do potrzeb użytkowników.

Kontrolery często zajmują się również zadaniem orkiestracji w aplikacji. Mogą one wywoływać różne modele lub operacje modelu w odpowiedniej kolejności, zarządzając w ten sposób zależnościami i przepływem danych między różnymi komponentami systemu. Ta rola orkiestracji jest niezbędna, gdy aplikacja wymaga skoordynowanego działania wielu modeli lub usług w celu realizacji złożonych zadań biznesowych.

Oprócz zarządzania przepływem informacji, kontrolery są odpowiedzialne za walidację danych wejściowych przed ich przekazaniem do modelu. Sprawdzają one poprawność formatu danych, ich kompletność oraz zgodność z oczekiwaniami biznesowymi. Poprzez wstępną walidację danych, kontrolery przyczyniają się do zwiększenia niezawodności i bezpieczeństwa aplikacji, ograniczając ryzyko wystąpienia błędów na etapie przetwarzania w modelu.

W paczce `controllers` utworzono trzy kontrolery, których zadaniem jest wykonywać stosowne dla żądań operacje. Wszystkie zostały zaimplementowane we frameworku Flask [79]. Do walidacji żądań wykorzystano bibliotekę Pydantic [86].

Kontroler `ItemController` odpowiada za zarządzanie cyklem życia przedmiotów w bazie danych. Zapewnia metody pozwalające na tworzenie, odczyt, aktualizację i usuwanie informacji o przedmiotach (CRUD). Obsługuje on żądania do kilku endpointów:

- `get_items` - obsługuje żądania GET na adresie `/api/items`, wykorzystując parametry zapytania `search` i `page` do filtrowania i stronicowania danych. Dane są pobierane metodą `get_all` z `ItemRepository` i zwracane w formacie JSON.
- `get_items_for_tags` - przetwarza żądania GET na adresie `/api/items/tags`, odbierając listę tagów i numer strony i używa metody `get_items_by_tags_sorted` do pobrania danych, które są zwracane jako JSON.
- `get_item_by_id` - służy do obsługi żądań GET dla konkretnego identyfikatora przedmiotu (`/api/items/<int:id>`). Jeśli przedmiot istnieje, jest zwracany w formacie JSON, w przeciwnym razie serwer odpowiada kodem 404.

- add_item - obsługuje żądania POST na adresie /api/items, gdzie nowy przedmiot jest tworzony na podstawie danych z ciała żądania i zwracany w formacie JSON. Funkcja ta również aktualizuje sekwencję identyfikatorów w bazie danych.
- update_item - obsługuje żądania PUT na adresie /api/items/<int:id>, umożliwiając aktualizację istniejącego przedmiotu. Po sprawdzeniu, czy przedmiot istnieje i zwróceniu 404 w przypadku negatywnego wyniku, przedmiot jest aktualizowany i zwracany jako JSON.
- delete_item - obsługuje żądania DELETE dla określonych identyfikatorów (/api/items/<int:id>), umożliwiając usunięcie przedmiotu. W przypadku braku przedmiotu zwracany jest kod 404, a po pomyślnym usunięciu kod 204, sygnalizujący sukces operacji bez zwracania treści.

Kontroler RatingController oferuje zarządzanie ocenami, włączając w to dodawanie, odczyt, aktualizację i usuwanie ocen.

- get_ratings - Ta funkcja obsługuje żądania GET do ścieżki /api/ratings, pobierając oceny z wykorzystaniem paginacji. Dane są zwracane w formacie JSON. Jest to podstawowa funkcja do odczytu danych o ocenach. Dodatkowo, RatingController implementuje cykliczne sprawdzanie potrzeby przeprowadzenia treningu modelu (funkcja check_training) oraz sam proces uczenia (run_ml_training), który jest inicjowany, gdy liczba nowych ocen przekroczy określony próg. Używa w tym celu wątków, co pozwala na nieblokujące przeprowadzanie operacji w tle.
- get_rating_by_id - obsługuje żądania GET na ścieżkę /api/ratings/<int:id>, zwracając szczegółowe informacje o konkretnej ocenie. Jeżeli dana ocena nie istnieje, zwracany jest kod błędu 404.
- add_rating - funkcja ta obsługuje żądania POST na /api/ratings, służące do dodawania nowych ocen. Przyjmuje identyfikatory użytkownika i przedmiotu oraz ocenę. Sprawdza, czy użytkownik i przedmiot istnieją i tworzy ocenę, zwracając jej dane w formacie JSON.
- update_rating - obsługuje żądania PUT na /api/ratings/<int:id>, umożliwiając aktualizację istniejącej oceny. Weryfikuje istnienie oceny, użytkownika oraz przedmiotu przed aktualizacją danych.
- delete_rating - ta funkcja obsługuje żądania DELETE na /api/ratings/<int:id>, umożliwiając usunięcie oceny. Jeżeli ocena nie istnieje, zwracany jest kod błędu 404, a po usunięciu kod 204, sygnalizujący pomyślne wykonanie operacji.
- get_training_status - funkcja ta dostarcza informacje o stanie procesu uczenia maszynowego, w tym, czy aktualnie trwa trening, ilość ocen od ostatniego treningu, a także czas ostatniego treningu. Informacje te są dostępne przez żądania GET na ścieżkę /api/ratings/status.

Kontroler UserController w aplikacji Flask zarządza operacjami związanymi z użytkownikami, korzystając z UserRepository do interakcji z bazą danych oraz z StrategyManager do zarządzania rekomendacjami.

- get_users - funkcja ta obsługuje żądania GET pod adresem /api/users, gdzie możliwe jest filtrowanie i stronicowanie listy użytkowników na podstawie przekazanych parametrów zapytania. Wszystkie dane użytkowników są zwracane w formacie JSON. Jest to podstawowa metoda do odczytywania danych o użytkownikach.
- get_user_by_id - obsługuje żądania GET pod /api/users/<int:id>, zwracając dane konkretnego użytkownika. Jeśli użytkownik nie zostanie znaleziony, zwracany jest status 404.

- `add_user` - ta funkcja obsługuje żądania POST pod `/api/users`, umożliwiając dodanie nowego użytkownika. Dane użytkownika są pobierane z ciała żądania i zapisywane w bazie danych, a następnie zwracane w formacie JSON.
- `update_user` - funkcja obsługuje żądania PUT pod `/api/users/<int:id>`, umożliwiając aktualizację danych istniejącego użytkownika. Jeżeli użytkownik o danym ID nie istnieje, zwracany jest błąd 404. W przeciwnym razie, dane są aktualizowane i zwracane w formacie JSON.
- `delete_user` - obsługuje żądania DELETE dla określonego ID użytkownika pod `/api/users/<int:id>`. Jeśli użytkownik o danym ID nie istnieje, zwracany jest błąd 404. W przypadku pomyślnego usunięcia, zwracany jest status 204.
- `get_recommendations` - funkcja ta obsługuje żądania GET pod `/api/users/<int:id>/recommendations`, gdzie na podstawie identyfikatora użytkownika generowane są rekomendacje. Używa `StrategyManager` do przewidywania rekomendacji na podstawie dostępnych strategii. Jeśli użytkownik nie istnieje lub nie ma dostępnych strategii, zwracany jest odpowiedni kod błędu.

5.5 Wstrzykiwanie zależności

Wstrzykiwanie zależności (ang. *Dependency Injection, DI)* to technika, która pozwala na wstrzykiwanie zależności obiektów do klasy zamiast pozwalać tej klasie na samodzielne tworzenie tych obiektów [85]. DI promuje luźne powiązania między klasami i ułatwia testowanie, ponieważ zależności mogą być łatwo zamieniane na mocki lub stuby. W kontekście aplikacji webowych, takich jak te oparte na Flasku, DI pozwala na wstrzykiwanie różnych komponentów (np. repozytoriów, serwisów) do kontrolerów.

W przypadku niniejszego systemu wstrzykiwanie zależności jest realizowane poprzez przekazywanie instancji repozytoriów i innych zależności do funkcji inicjalizujących kontrolery. Zależności te są tworzone na poziomie głównego modułu aplikacji, a następnie przekazywane do odpowiednich funkcji jako argumenty. Funkcje te wykorzystują przekazane zależności do konfigurowania endpointów i logiki aplikacji.

Przykładem może być kontroler `RatingController`, którego fragment pokazano w kodzie 5.3, który zarządza operacjami związanymi z ocenami użytkowników. Funkcja `initialize_rating_controller` przyjmuje instancje `RatingRepository`, `UserRepository`, `ItemRepository` oraz `StrategyManager` jako argumenty. Te instancje są następnie używane wewnątrz funkcji obsługujących różne endpointy.

Dependency Injection w tym przypadku pozwala na centralizację zarządzania zależnościami, co znacząco ułatwia testowanie i utrzymanie kodu. Funkcje kontrolera mogą skoncentrować się na swojej logice, nie martwiąc się o tworzenie czy konfigurację zależnych obiektów. Dzięki temu, gdy zajdzie potrzeba zmiany implementacji którejkolwiek z zależności, można to zrobić w jednym miejscu, bez konieczności modyfikowania kodu kontrolera.

Kod 5.3: Fragment kontrolera ocen

```
def initialize_rating_controller(
    app,
    rating_repository: RatingRepository, user_repository: UserRepository,
    item_repository: ItemRepository,
    strategy_manager: StrategyManager,
    train_every: int = 3,
    train_on_start: bool = True,
):
    last_count = rating_repository.count()
```

```
is_training = False
last_training = 0

@app.route('/api/ratings', methods=['GET'])
@validate_request_data(GetAllRatingsRequest, source='query')
def get_ratings():
    page = int(request.args.get('page'))
    ratings = rating_repository.get_all(page)
    return jsonify([rating.__dict__ for rating in ratings])
```

Na przykład, funkcja `get_ratings` w `initialize_rating_controller` pokazanym w kodzie 5.3 obsługuje żądania GET na adres `/api/ratings`, używając `rating_repository` do pobrania ocen z bazy danych. Dzięki DI, `rating_repository` jest dostarczany jako argument funkcji inicjalizującej, co ułatwia jego podmianę na inną implementację lub mock podczas testowania.

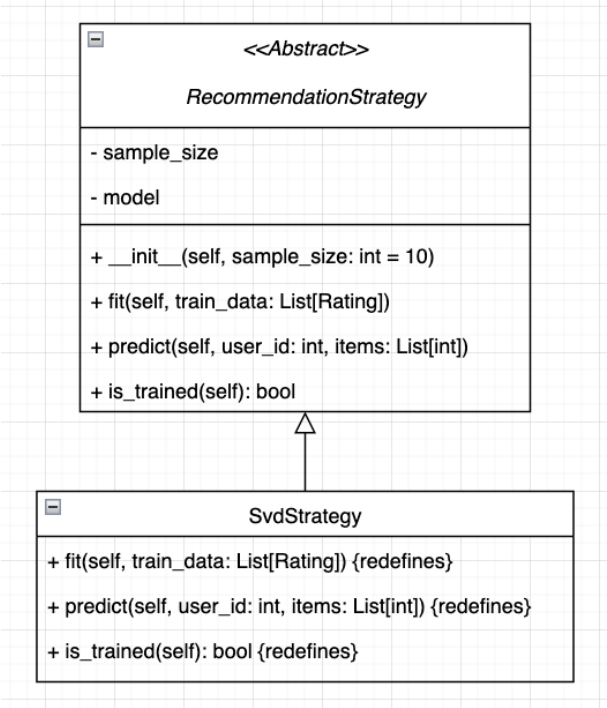
5.6 Modułowe ładowanie strategii

System rekomendacji treści zaprezentowany w tej publikacji opiera się na architekturze wtyczek i modułowym ładowaniu strategii rekomendacji. Takie podejście gwarantuje dużą elastyczność w zarządzaniu zróżnicowanymi algorytmami rekomendacji, bez potrzeby modyfikowania głównego kodu systemu. Poszczególne strategie, zaimplementowane jako niezależne moduły, mogą być dynamicznie ładowane, aktualizowane lub dezaktywowane podczas pracy aplikacji. Ponadto, taka struktura systemu pozwala na testowanie wtyczek w odizolowanym środowisku, co w znacznym stopniu usprawnia proces identyfikacji i eliminacji potencjalnych błędów.

Wszystkie ładowane moduły implementują abstrakcyjną klasę bazową `RecommendationStrategy`. Ta klasa definiuje ujednolicony interfejs dla wszystkich strategii rekomendacji, zawierając abstrakcyjne metody takie jak fit, predict i is_trained. Jest to komponent systemu, który zapewnia jednolitość i przewidywalność działania różnych algorytmów rekomendacyjnych.

- Metoda `fit` służy do trenowania modelu na podstawie dostarczanych danych. Każda implementacja tej klasy musi zdefiniować, jak dane uczące są przetwarzane.
- Metoda `predict` jest odpowiedzialna za generowanie rekomendacji. Bazując na identyfikatorze użytkownika oraz liście przedmiotów podlegających ocenie zwraca najtrafniejsze przedmioty.
- Metoda `is_trained` zwraca informację czy model został już wytrenowany.

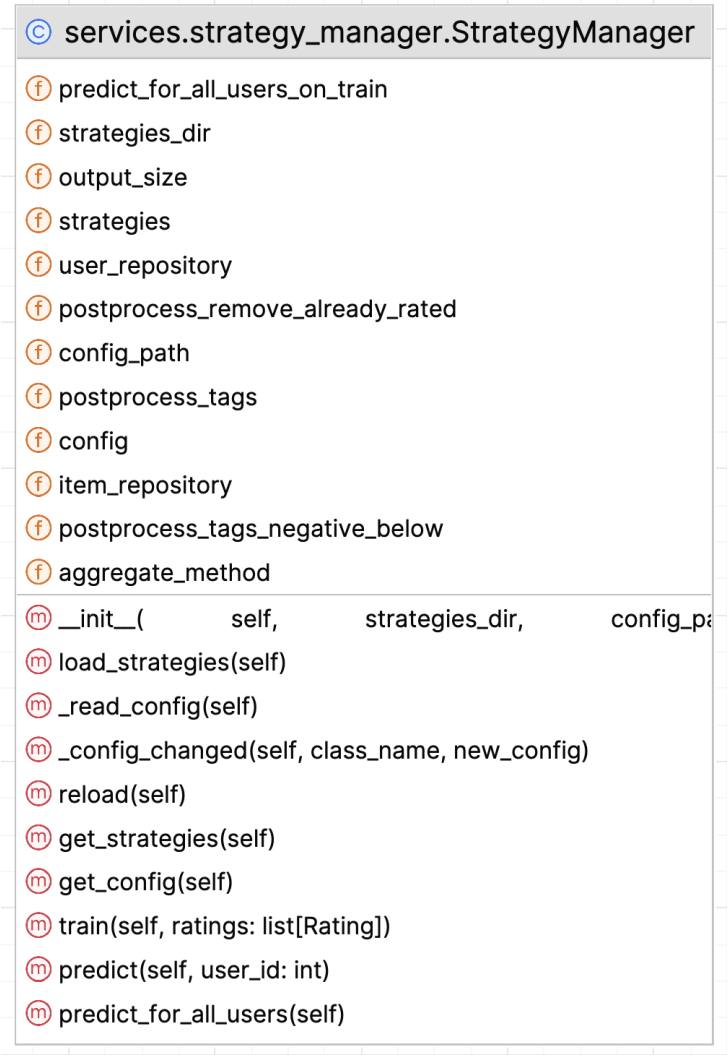
Na rysunku 5.5 przedstawiono klasę abstrakcyjną `RecommendationStrategy` i jedną z implementacji, SvdStrategy.



Rysunek 5.5: Diagram klasy abstrakcyjnej RecommendationStrategy i jej implementacji SvdStrategy

Kluczową rolę w zarządzaniu modułami pełni usługa `StrategyManager`. Jej głównym zadaniem jest automatyzacja procesu integracji i koordynacji działania różnorodnych algorytmów rekomendacyjnych, implementowanych jako oddzielne moduły wtyczek. Dzięki `StrategyManager` system jest w stanie dynamicznie reagować na zmiany w wymaganiach i preferencjach użytkowników, adaptując dostępne algorytmy bez konieczności interwencji w kodzie lub restartowania całego systemu.

Na poziomie operacyjnym, `StrategyManager` odpowiada za wczytanie konfiguracji z pliku JSON, co pozwala na określenie, które strategie są aktywne, jakie mają parametry, oraz czy wymagają one odświeżenia lub ponownego załadowania. Po załadowaniu odpowiednich modułów strategii, menedżer zarządza ich cyklem życia, inicjując procesy trenowania na podstawie dostarczonych danych treningowych oraz wywołując metody predykcji, kiedy generowanie rekomendacji jest wymagane. Ponadto, `StrategyManager` oferuje funkcjonalność związaną z post-przetwarzaniem (ang. `postprocessing`) rekomendacji, co obejmuje agregację wyników z różnych strategii i filtrowanie rekomendacji, aby unikać sugerowania użytkownikowi treści, które już ocenił. Na rysunku 5.6 pokazano diagram klasy `StrategyManager`.



Rysunek 5.6: Diagram klasy StrategyManager

Konstruktor klasy `StrategyManager` inicjuje ścieżkę do katalogu ze strategiami, ścieżkę do pliku konfiguracyjnego oraz różne parametry takie jak repozytoria dla użytkowników i przedmiotów oraz opcje postprocesowania. Dane te mogą być konfigurowalne za pomocą zmiennych środowiskowych i są przekazywane w momencie tworzenia instancji obiektu przy uruchomieniu systemu.

Dynamiczne ładowanie modułów strategii zrealizowane zostało z wykorzystaniem wbudowanej w Python biblioteki `importlib`. Wczytywanie i tworzenie nowych instancji strategii oparte zostało na analizie plików w określonym katalogu oraz sprawdzeniu, czy konkretna strategia została włączona i dostępna w pliku konfiguracyjnym.

W przypadku zmiany konfiguracji w czasie działania programu, system wykryje ją za pomocą metody `_config_changed` i załaduje do pamięci nową konfigurację w miejsce starej. W ten sposób nie jest wymagane ponowne uruchomienie systemu, aby zmienić parametry lub włączyć/wyłączyć daną strategię rekomendacyjną.

Rekomendacje mogą być uruchomione na żądanie dla wyłącznie jednego użytkownika lub dla wszystkich. W tym celu ustawiana jest flaga `predict_for_all_users_on_train`. Proces przewidywania będzie trwał dłużej, jednak w przypadku gdy rekomendacje dla danego użytkownika będą już obliczone to wyniki będą dostępne natychmiast z uwagi na ich zapisywanie w bazie danych w formie cache.

Na wynik finalnej rekomendacji może wpłynąć wiele modeli jednocześnie. Ich udział w procesie

może zostać ustawiony w pliku konfiguracyjnym, a metoda agregacji wyników z różnych modeli może być określona za pomocą zmiennej środowiskowej AGGREGATE_METHOD. Ilość wynikowych rekomendacji może być ustawiona przez zmienną OUTPUT_SIZE.

Ponadto, jeśli proces postprocessingu został zainicjowany za pomocą zmiennej środowiskowej POST-PROCESS_TAGS, sekwencja rekomendacji może ulec transformacji. Pierwotna pula rekomendacji, wygenerowana przez modele uczenia maszynowego, zostanie posortowana z wykorzystaniem modelu statystycznego, który, w przypadku wystąpienia tagów w rekomendowanych treściach, poszukuje korelacji między nimi i historycznymi ocenami wystawionymi przez użytkownika. Dodatkowo, podczas postprocessingu, możliwe jest skonfigurowanie systemu tak, aby odrzucał treści poniżej predefiniowanego progu ocen, wykorzystując zmienną środowiskową POSTPROCESS_TAGS_NEGATIVE_BELOW. Zmienna POST-PROCESS_REMOVE_ALREADY_RATED natomiast, informuje system, aby usunął z finalnego zbioru pozycje, które zostały już poddane ocenie przez danego użytkownika.

5.7 Ładowanie danych testowych

W celu przetestowania systemu rekomendacji na rzeczywistych danych, wykorzystano zbiór danych MovieLens [46]. Zbiór ten zawiera oceny filmów udzielone przez użytkowników na platformie MovieLens. Do załadowania danych do bazy danych PostgreSQL opracowano dedykowany skrypt.

Skrypt rozpoczyna od nawiązania połączenia z bazą danych przy użyciu Psycopg [27] i parametrów połączenia zdefiniowanych w zmiennych środowiskowych. Następnie wywoływana jest funkcja run_-migration, odpowiedzialna za wykonanie migracji schematu bazy danych.

W dalszej części skryptu tworzone są instancje repozytoriów dla tabel users, ratings i items. Repozytoria te udostępniają metody do tworzenia rekordów w odpowiednich tabelach.

Proces ładowania danych rozpoczyna się od odczytu pliku CSV zawierającego informacje o filmach. Dla każdego wiersza w pliku, wyodrębniane są wartości item_id, title i genres. Identyfikator filmu item_id jest konwertowany na typ całkowity. Gatunki filmowe zapisane w kolumnie genres są dzielone na podstawie separatora " i zapisywane w tablicy tags. Następnie wywoływana jest metoda create_-with_id repozytorium item_repository, która tworzy nowy rekord w tabeli items z podanymi wartościami.

Kolejnym krokiem jest ładowanie danych o ocenach użytkowników z pliku CSV. Dla każdego wiersza w pliku, wyodrębniane są wartości user_id, item_id i rating. Identyfikatory użytkownika i filmu są konwertowane na typ całkowity, a ocena na typ zmiennoprzecinkowy. Przed zapisaniem oceny, sprawdzane jest, czy użytkownik o danym identyfikatorze już istnieje w tabeli users. Jeśli nie, tworzony jest nowy rekord użytkownika za pomocą metody create_with_id repozytorium user_repository. Następnie, wywoływana jest metoda create repozytorium rating_repository, która tworzy nowy rekord oceny w tabeli ratings.

Po zakończeniu ładowania danych, wywoływane są metody fix_id_sequence dla repozytoriów user_repository i rating_repository, które aktualizują sekwencje identyfikatorów w bazie danych.

5.8 Testowanie

W celu zapewnienia wysokiej jakości kodu oraz poprawności działania aplikacji, przygotowano zestaw testów jednostkowych z wykorzystaniem bibliotek unittest oraz unittest.mock [9]. Testy zaprojektowano w taki sposób, aby symulować interakcję z aplikacją Flask oraz weryfikować poprawność działania endpointów API zdefiniowanych w kontrolerach. Do odizolowania testów od rzeczywistej implementacji repozytoriów, zastosowano koncepcję mocków (ang. *mocks*) z biblioteki unittest.mock. W ramach zestawu testowego dla kontrolera ItemController, w metodzie setUp tworzony jest mock ItemRepository z wykorzystaniem klasy MagicMock. Klasa ta pozwala na precyzyjne kontrolowanie zachowania metod repozytorium oraz weryfikację ich wywołań. Mock ten jest następnie przekazywany do

inicjalizacji kontrolera, co umożliwia testowanie jego działania bez konieczności interakcji z rzeczywistą bazą danych.

Przygotowane testy obejmują scenariusze, takie jak zwracanie listy obiektów `Item`, pojedynczego obiektu `Item` lub wartości `None` przez metody repozytorium. Dla każdego testowanego endpointu weryfikowany jest status odpowiedzi HTTP oraz poprawność struktury i zawartości zwracanego JSON-a. Ponadto, przy użyciu asercji `assert_called_once_with`, sprawdzane jest, czy odpowiednie metody repozytorium zostały wywołane z oczekiwanymi argumentami. Wykorzystano klasę `MagicMock` do tworzenia mocków repozytoriów z określonymi parametrami `spec`. Parametr ten pozwala na precyzyjną kontrolę nad zachowaniem metod repozytorium oraz weryfikację ich wywołań. Dzięki temu mechanizmowi możliwe jest symulowanie różnych scenariuszy, takich jak zwracanie określonych wartości przez metody repozytorium, co umożliwia dokładne przetestowanie logiki biznesowej kontrolera.

W testach jednostkowych dla kontrolera `UserController`, oprócz wykorzystania bibliotek `unittest` i `unittest.mock`, zastosowano również mocki dla dodatkowych zależności, takich jak `StrategyManager`. Pozwala to na odizolowanie testów od rzeczywistej implementacji tych zależności i skupienie się na testowaniu samego kontrolera. Testy weryfikują nie tylko operacje CRUD na użytkownikach, ale również działanie endpointu odpowiedzialnego za pobieranie rekomendacji dla danego użytkownika.

W testach kontrolera `RatingController` wprowadzono dodatkową flagę `check_training_flag`, która pozwala kontrolować cykliczne wywołania w tle funkcji `check_training`, służącej do sprawdzenia statusu treningu modelu rekomendacji. Flaga ta umożliwia zatrzymanie cyklicznych wywołań podczas testów, co jest niezbędne, aby testy mogły zakończyć się poprawnie bez konieczności wymuszania zamknięcia procesu.

Poza testowaniem kontrolerów, poddano testom także moduły strategii rekomendacji. Celem tych testów jest weryfikacja poprawności działania poszczególnych funkcji i metod odpowiedzialnych za generowanie rekomendacji.

W ramach klasy `TestGetTopNRecommendations` przeprowadzono testy jednostkowe dla funkcji `get_top_n_recommendations`, która jest odpowiedzialna za selekcję n najlepszych rekomendacji na podstawie przewidywań. Testy weryfikują, czy funkcja zwraca oczekiwane wyniki dla różnych wartości parametru n oraz czy liczba zwracanych rekomendacji jest ograniczona do zadanej wartości. Zastosowano asercje `assertEqual` oraz `assertLessEqual`, aby porównać wyniki zwrócone przez funkcję z oczekiwanymi wartościami.

Klasa `TestSvdStrategy` testuje moduł *SvdStrategy*. Analogiczne testy zostały przygotowane również dla pozostałych modułów strategii rekomendacji. Przykładowy test jednostkowy dla modułu SVD został pokazany w kodzie 5.4.

Kod 5.4: Fragment zestawu testowego dla modułu SVD

```
class TestSvdStrategy(unittest.TestCase):
    @patch('builtins.open', new_callable=mock_open)
    @patch('os.remove')
    @patch('surprise.Dataset.load_from_file')
    @patch('plugins.svd_strategy.SVD')
    def test_fit(self, MockSVD, MockLoadFromFile, mock_remove, mock_open):
    train_data = [
        Rating(id=1, user_id=1, item_id=1, rating=5),
        Rating(id=2, user_id=1, item_id=2, rating=3)
    ]

    mock_model = MockSVD.return_value
    mock_model.fit = MagicMock()

    mock_data = MagicMock()
```

```
mock_data.build_full_trainset = MagicMock()
MockLoadFromFile.return_value = mock_data

strategy = SvdStrategy(sample_size=1)
strategy.fit(train_data)

mock_open.assert_called_once()
mock_remove.assert_called_once()
MockSVD.assert_called_once()
MockLoadFromFile.assert_called_once()
mock_model.fit.assert_called_once()
```

Test `test_fit` z `TestSvdStrategy` weryfikuje proces trenowania modelu SVD z wykorzystaniem mocków i asercji. Zastosowano dekoratory `@patch` do mockowania funkcji i metod, takich jak `open`, `os.remove`, `surprise.Dataset.load_from_file` oraz `plugins.svd_strategy.SVD`. Dzięki temu możliwe jest kontrolowanie zachowania tych zależności podczas testów. Wykorzystano asercje `assert_called_once` oraz `assert_called_once_with`, aby zweryfikować, czy odpowiednie funkcje i metody zostały wywołane z oczekiwanymi argumentami.

Test `test_predict` sprawdza działanie metody `predict`, która jest odpowiedzialna za generowanie rekomendacji dla danego użytkownika i listy przedmiotów. W teście wykorzystano mocki do symulowania zachowania modelu SVD i weryfikowano poprawność zwracanych wyników oraz typ zwracanej wartości. Zastosowano asercje `assertTrue` oraz `assertEqual`, aby sprawdzić, czy wyniki są zgodne z oczekiwaniami.

Test `test_is_trained` weryfikuje działanie metody `is_trained`, która informuje, czy model został wytrenowany. Sprawdzane jest zachowanie metody w przypadku, gdy model nie jest zdefiniowany oraz gdy jest zdefiniowany. Wykorzystano asercje `assertFalse` oraz `assertTrue` do weryfikacji poprawności zwracanych wartości logicznych.

Natomiast test `test_predict_not_trained` sprawdza zachowanie metody `predict` w przypadku, gdy model nie został wcześniej wytrenowany. Oczekiwane jest, że metoda zwróci pustą listę, co jest weryfikowane za pomocą asercji `assertEqual`.

5.9 Obraz Dockera

Przygotowany prototyp systemu rekomendacji treści został udostępniony na platformie DockerHub [31] w formie obrazu dla programu *Docker* pod nazwą `pilotpirxie/recommendation`. Proces konteneryzacji został oparty o plik *Dockerfile* przedstawiony w kodzie 5.5, który reprezentuje warstwy i kolejne kroki budowy oraz uruchomienia obrazu.

Kod 5.5: Zawartość pliku Dockerfile

```
FROM python:3.10-slim AS build

WORKDIR /app

RUN apt-get update && apt-get install -y \
gcc \
build-essential \
&& rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
```

```
RUN pip install --no-cache-dir -r requirements.txt

FROM python:3.10-slim

COPY --from=build /usr/local/lib/python3.10/site-packages /usr/local/lib/python3.10/
    site-packages
COPY --from=build /usr/local/bin /usr/local/bin

COPY src /app/src

WORKDIR /app/src

ENV PORT=3000
ENV VERBOSE=true

EXPOSE 3000

CMD ["python", "-m", "app"]
```

W pierwszym kroku, przy pomocy komendy `FROM` wybierany jest bazowy obraz, który zawiera system operacyjny, podstawowe biblioteki i programy w tym SDK Python 3.10 [4]. W tym przypadku używamy obrazu `python:3.10-slim`, co oznacza, że podstawą naszego procesu budowy będzie okrojone i przez to mniej zasobożerne środowisko zajmujące jedynie 46.17 MB po skompresowaniu w porównaniu z obrazem `python:3.10`, którego skompresowany obraz zajmuje 358.2 MB. Podczas pracy nad aplikacją przeprowadzono również podejścia do uruchomienia jej w oparciu o system Alpine, który słynnie z niezwykle niskich rozmiarów i mniejszego narzutu na tworzone obrazy [1]. Próbę podjęto wykorzystując obraz `python:3.10-alpine` o rozmiarze skompresowanym 18.49 MB jednak z uwagi na przejściową zależność *scipy*, która jest wymagana przez *scikit-learn*, który to z kolei jest wymagany przez *surprise* nie udało się zainstalować zależności. System Alpine nie zawierał domyślnie wymaganych bibliotek, a dodanie ich w kolejnych krokach sprawiało znaczne wydłużenie czasu budowy obrazu. Obraz *slim* zawiera system, niekoniecznie Alpine, który podobnie jak wcześniej wspomniany został przygotowany aby być mały i wydajny. W jego przypadku instalacja zależności była znacznie prostsza, szybsza i wymagała instalacji zaledwie kilku dodatkowych bibliotek.

W kolejnym kroku ustawiono katalog roboczy na `/app`, w którym będą wykonywane kolejne kroki budowania. Za pomocą komendy `apt-get` zaktualizowano listę pakietów systemowych oraz zainstalowano wymagane narzędzia do kompilacji, takie jak `gcc` i `build-essential`. Po instalacji usunięto niepotrzebne pliki z listami pakietów, aby zmniejszyć rozmiar obrazu. Następnie kopiowany jest plik z zależnościami `requirements.txt` i za pomocą `pip` następuje ich instalacja z flagą wyłączającą cache.

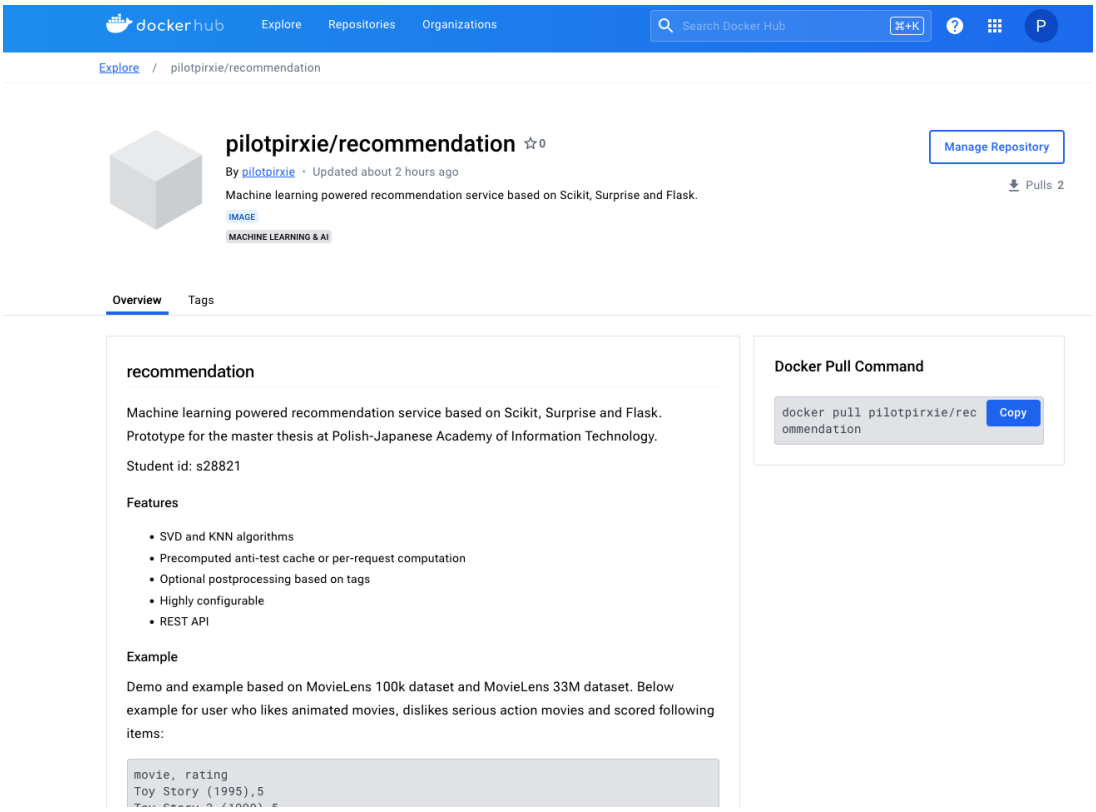
Zastosowano wieloetapowy proces budowy obrazu w celu redukcji jego finalnego rozmiaru. Poprzednio opisane kroki dotyczyły pierwszego etapu. W następnym etapie po raz kolejny obraz bazowy. Również jest to `python:3.10-slim`. Tym razem definiuje on bazę dla finalnego kontenera, który zostanie utworzony za pomocą tak przygotowanego obrazu.

Następnie skopiowano zależności i pliki binarne skompilowane w poprzednim etapie do finalnego obrazu. Również kod źródłowy aplikacji został skopiowany z lokalnego katalogu `src` do `/app/src` w kontenerze.

Przed uruchomieniem aplikacji ustawiane są dwie zmienne środowiskowe: `PORT=3000` oraz `VERBOSE=true`. Dzięki nim serwer uruchomi się na porcie lokalnym 3000 wewnątrz kontenera oraz będzie wypisywał do standardowego wyjścia więcej informacji o swoim działaniu. Za pomocą komendy `EXPOSE` udostępniono lokalny port 3000 aplikacji na zewnątrz kontenera. W ostatnim kroku uruchamiana jest aplikacja za pomocą `CMD` z argumentami `python -m app`.

Po przygotowaniu obrazu został on wysłany na DockerHub z tagiem `latest`. Na rysunku 5.7 przed-

stawiono zrzut ekranu ze strony obrazu.



Rysunek 5.7: Strona obrazu na DockerHub

Tak przygotowany obraz może być użyty do uruchomienia serwera przez inne osoby bez konieczności ręcznego konfigurowania środowiska Python 3. Zawiera on wszystkie niezbędne zależności i kod źródłowy, gotowy do uruchomienia w środowisku produkcyjnym.

6. Weryfikacja wyników

W niniejszym rozdziale opisano proces weryfikacji modułów systemu rekomendacji treści wraz z oceną ich efektywności, wykorzystując metryki błędu RMSE (ang. *Root Mean Square Error*) i MAE (ang. *Mean Absolute Error*) [48]. Zaprezentowano także przykład praktycznego wdrożenia systemu na obszernym zbiorze danych MovieLens 33M [46]. Ponadto, przeprowadzono analizę zużycia zasobów obliczeniowych, w tym czasu przetwarzania oraz wykorzystania pamięci RAM i mocy obliczeniowej CPU, umożliwiając ocenę wydajności i skalowalności zaproponowanego systemu w kontekście obsługi dużych zbiorów danych.

6.1 Skuteczność modeli

Na potrzeby niniejszej pracy przygotowano moduły generowania rekomendacji w oparciu o dwa modele maszynowe SVD i KNNBasic. Obydwa w implementacji pochodzącej z biblioteki Surprise [101]. Ich skuteczność została zbadana przy pomocy dwóch metryk pierwiastka średniokwadratowego błędu (ang. *Root Mean Square Error, RMSE*) i średniego błędu bezwzględnego (ang. *Mean Absolute Error, MAE*).

Pierwiastek błędu średniokwadratowego to miara, która pozwala ocenić, jak dokładnie systemy rekomendacji przewidują preferencje użytkowników. Miara ta pokazuje różnicę między ocenami, które system przewidział i ocenami, które użytkownicy faktycznie wystawili. Im mniejsza wartość RMSE, tym lepiej system radzi sobie z przewidywaniem, co spodoba się użytkownikom. RMSE jest szczególnie czuły na duże rozbieżności między przewidywanymi i rzeczywistymi ocenami. Dzięki temu łatwo można wychwycić poważne błędy w rekomendacjach. System jest *karany* za duże pomyłki w przewidywaniach, co ma kluczowe znaczenie w przypadku systemów rekomendacji. Duże rozbieżności między przewidywaniami i rzeczywistością mogą bowiem prowadzić do znacznego niezadowolenia użytkowników.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6.1}$$

gdzie:

- n to liczba próbek w zbiorze danych
- y_i to rzeczywista wartość (ocena lub preferencja) dla i-tej próbki
- $\hat{y}_i$ to przewidywana wartość (ocena lub preferencja) dla i-tej próbki
- $(y_i - \hat{y}_i)^2$ to kwadrat różnicy między rzeczywistą i przewidywaną wartością dla i-tej próbki

Średni błąd bezwzględny, to kolejna metoda oceny dokładności modeli. W przypadku modeli rekomendacji polega na obliczeniu średniej wartości bezwzględnej różnicy pomiędzy ocenami lub preferencjami przewidywanymi przez system i rzeczywistymi ocenami lub preferencjami wyrażonymi przez użytkowników. Im niższa wartość MAE, tym lepiej system radzi sobie z trafnym przewidywaniem preferencji użytkowników. Zaletą MAE jest to, że wszystkie błędy traktowane są jednakowo, bez względu na to, czy są one duże czy małe. Dzięki temu otrzymujemy ogólny obraz jakości rekomendacji generowanych przez system. Metryka ta może być szczególnie przydatna w sytuacjach, gdy zależy nam na równomiernej ocenie wszystkich rozbieżności między przewidywaniami i rzeczywistością. Stosowanie MAE jako miary jakości rekomendacji może być wskazane w przypadkach, gdy system powinien działać w sposób bardziej zachowawczy i mniej ryzykowny. Wynika to z faktu, iż MAE nie przywiązuje szczególnej wagi do sporadycznych błędów o dużej skali, koncentrując się raczej na ogólnej zgodności przewidywań z preferencjami użytkowników.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{6.2}$$

gdzie:

- n to liczba próbek w zbiorze danych
- y_i to rzeczywista wartość (ocena lub preferencja) dla i-tej próbki
- $\hat{y}_i$ to przewidywana wartość (ocena lub preferencja) dla i-tej próbki
- $|y_i - \hat{y}_i|$ to wartość bezwzględna różnicy między rzeczywistą i przewidywaną wartością dla i-tej próbki

W przypadku zastosowanych modeli SVD i KNNBasic z biblioteki Surprise, ich wartości MAE i RMSE zostały ocenione na zbiorze danych udostępnionym przez MovieLens. Zbiór ten i jego historia zostały opisane w innej części niniejszej pracy. Na potrzeby testów wykorzystano wbudowany podzbiór `ml-100k`, który składa się z 100 tysięcy losowo wybranych ocen z pełnego zbioru. W kodzie 6.1 przedstawiono kod wykorzystany do oceny modeli.

Kod 6.1: Ocena modeli KNNBasic i SVD

```
from surprise import SVD, Dataset, Reader
from surprise import accuracy
from surprise.model_selection import cross_validate
from surprise import KNNBasic, BaselineOnly, SVDpp
import os

data = Dataset.load_builtin('ml-100k')

algorithms = {
    'SVD': SVD(),
    'KNN': KNNBasic(),
}

results = {}

for name, algo in algorithms.items():
    results[name] = cross_validate(algo, data, measures=['RMSE', 'MAE'], cv=5, verbose=
        True)

for name, result in results.items():
    print(f"Results for {name}:")
    print(f"Mean RMSE: {sum(result['test_rmse']) / len(result['test_rmse'])}")
    print(f"Mean MAE: {sum(result['test_mae']) / len(result['test_mae'])}")
    print()
```

Ocenę modelu zrealizowano z wykorzystaniem metody walidacji krzyżowej (ang. *cross validation*) [18]. Metoda ta polega na podzieleniu dostępnego zbioru danych na kilka rozłącznych podzbiorów, a następnie iteracyjnym trenowaniu i testowaniu modelu na różnych konfiguracjach tych podzbiorów. Celem tej techniki jest ocena zdolności generalizacyjnych modelu, czyli jego wydajności na danych, które nie były wykorzystane podczas procesu uczenia. Dzięki temu można zidentyfikować i ograniczyć

ryzyko przeuczenia (ang. *overfitting*), które objawia się zbytnią specjalizacją modelu względem danych treningowych i słabą skutecznością na nowych, nieznanych obserwacjach.

W zaprezentowanym fragmencie kodu 6.1 zastosowano funkcję `cross_validate` pochodzącą z biblioteki Surprise. Ustawienie parametru `cv=5` dzieli cały zbiór danych na 5 równych części, zwanych foldami. Proces walidacji krzyżowej składa się z 5 iteracji, w których każdy fold kolejno pełni rolę zbioru testowego, podczas gdy pozostałe foldy są łączone w zbiór treningowy. W ramach pojedynczej iteracji model jest trenowany na zbiorze treningowym, a następnie ewaluowany na zbiorze testowym w oparciu o zdefiniowane miary jakości. Finalnie wyniki dla SVD przedstawiono w tabeli 6.1 i dla KNNBasic w tabeli 6.2.

Tabela 6.1: Wyniki walidacji krzyżowej algorytmu SVD

Metryka	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Średnia**	**Std**
RMSE (testset)	0.9421	0.9370	0.9302	0.9358	0.9396	0.9370	0.0040
MAE (testset)	0.7412	0.7348	0.7344	0.7394	0.7396	0.7379	0.0028
Czas treningu	0.32	0.31	0.32	0.31	0.31	0.31	0.00
Czas testu	0.05	0.03	0.05	0.05	0.05	0.05	0.01

Tabela 6.2: Wyniki walidacji krzyżowej algorytmu KNNBasic

Metryka	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Średnia**	**Std**
RMSE (testset)	0.9752	0.9795	0.9771	0.9790	0.9810	0.9784	0.0020
MAE (testset)	0.7698	0.7752	0.7710	0.7726	0.7753	0.7728	0.0022
Czas treningu	0.06	0.07	0.07	0.07	0.07	0.07	0.00
Czas testu	0.75	0.77	0.76	0.75	0.76	0.76	0.01

Wynik działania walidacji krzyżowej dla modeli SVD i KNNBasic wskazuje, że algorytm SVD osiąga lepsze wyniki w zakresie dokładności przewidywań, co jest widoczne przez niższe wartości RMSE i MAE odpowiednio o 4,23% i 4,51%. Zaobserwowana różnica w dokładności sugeruje, iż zastosowanie techniki redukcji wymiarowości, leżącej u podstaw działania SVD, pozwala na efektywniejsze wyodrębnienie istotnych wzorców z analizowanych danych.

Warto jednak zwrócić uwagę na fakt, iż model KNNBasic charakteryzuje się krótszym czasem procesu uczenia, co jest konsekwencją specyfiki działania tego algorytmu. W przeciwieństwie do SVD, KNNBasic nie przeprowadza kompleksowych obliczeń na etapie trenowania, a jedynie w momencie generowania rekomendacji dla konkretnego użytkownika. Ta właściwość może okazać się korzystna w systemach, które wymagają częstej aktualizacji danych treningowych lub operują na bardzo obszernych zbiorach danych. Niemniej jednak, jak pokazują wyniki przeprowadzonej walidacji krzyżowej, wyższa efektywność obliczeniowa na etapie uczenia wiąże się z niższą jakością generowanych rekomendacji oraz dłuższym czasem ich generowania w przypadku algorytmu KNNBasic.

6.2 Przykładowa integracja systemu

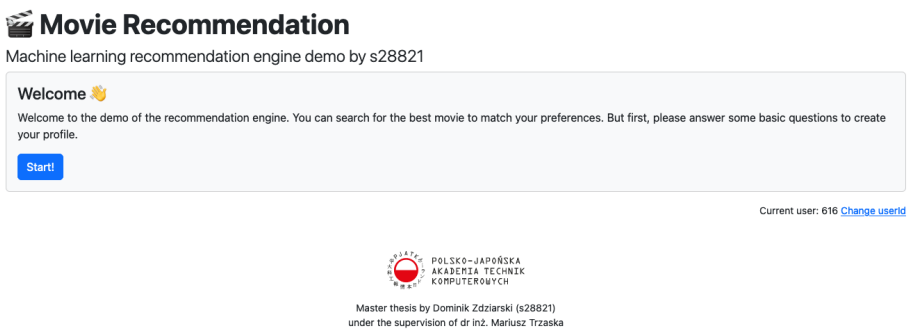
Poza syntetycznymi wynikami, na potrzeby pracy dyplomowej przygotowano również aplikację kliencką wyszukiwarki polecanych filmów, z którą zintegrowany został system rekomendacji. Tematyka tej aplikacji to rekomendacje filmów w oparciu o zestaw danych MovieLens [46]. Aplikacja krok po kroku od momentu uruchomienia przeprowadza użytkownika przez wstępny wybór filmów by ocenić ogólne preferencje oglądającego, rozwiązując w ten sposób problem zimnego startu opisany w innej części niniejszej pracy. W następnym kroku aplikacja wyświetla wstępnie dobrane filmy bazując na etykietach przypisanych do tematyki, nastroju i okazji. Kolejne rekomendacje są już wyświetlane w oparciu o wybrane moduły z modelami uczenia maszynowego w systemie.

Aplikacja internetowa została zaimplementowana z wykorzystaniem biblioteki JavaScript o nazwie React [5], która umożliwia tworzenie interaktywnych interfejsów użytkownika. Kod źródłowy aplikacji został napisany w języku TypeScript [8], będącym nadzbiorem standardowego języka JavaScript o statyczne typowanie, co pozwala na wykrywanie błędów na etapie kompilacji i poprawia czytelność oraz utrzymywalność kodu.

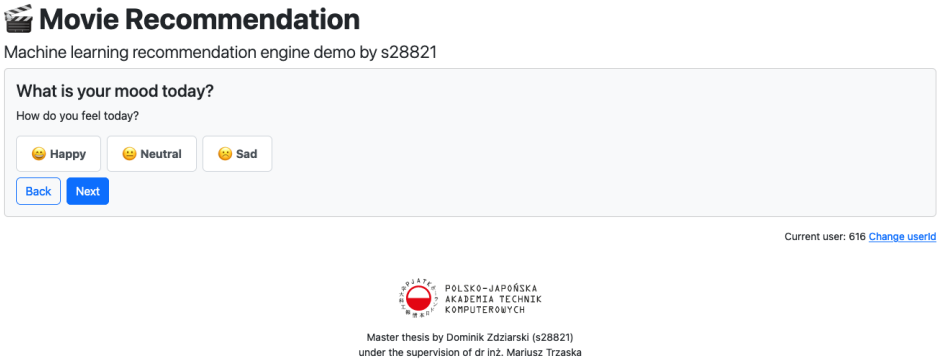
W procesie budowania aplikacji zastosowano bundler Vite [10]. Jego głównym zadaniem jest przetwarzanie i optymalizacja plików źródłowych, takich jak pliki TypeScript, CSS oraz innych zasoby, a następnie generowanie finalnego pakietu, który może być serwowany i uruchamiany w przeglądarkach internetowych. Praca z Vite na potrzeby niniejszej pracy charakteryzowała się szybkością działania i minimalną konfiguracją, co pozwoliło rozpocząć pracę bez poświęcania dodatkowego czasu na samo środowisko developerskie.

W celu zapewnienia responsywności i atrakcyjnego wyglądu aplikacji, zdecydowano się na wykorzystanie popularnego frameworka kaskadowych arkuszy styli o nazwie Bootstrap 5 [2]. Bootstrap oferuje zestaw predefiniowanych styli i komponentów interfejsu użytkownika, takich jak responsywne układy elementów, przyciski, pola edycji, style fontów, kolory dla różnych elementów, formularze czy elementy nawigacyjne.

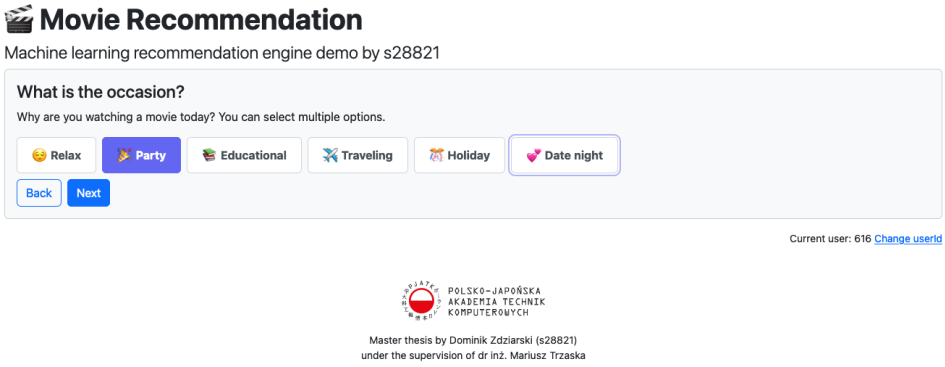
Na rysunkach 6.1, 6.2, 6.3, 6.4 przedstawiono kolejne kroki wstępnego wyboru rekomendacji dla użytkownika, który nie ocenił wcześniej żadnych filmów.



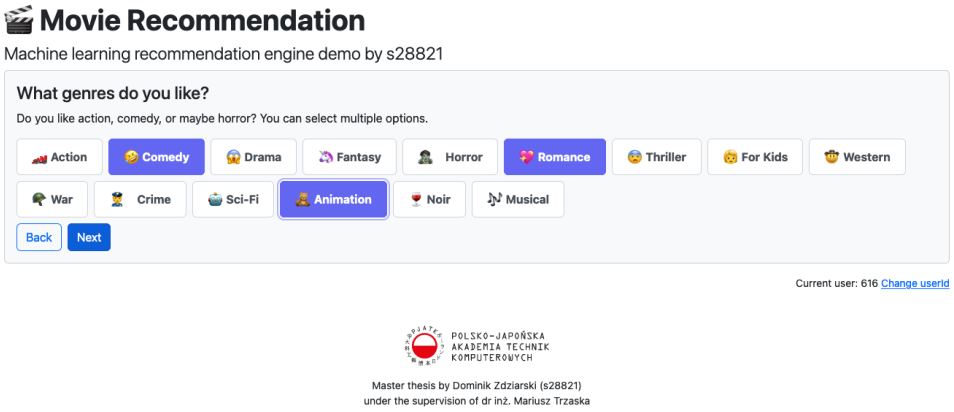
Rysunek 6.1: Zrzut ekranu prezentujący pierwszy widok wyszukiwarki polecanych filmów



Rysunek 6.2: Zrzut ekranu prezentujący krok z wyborem nastroju

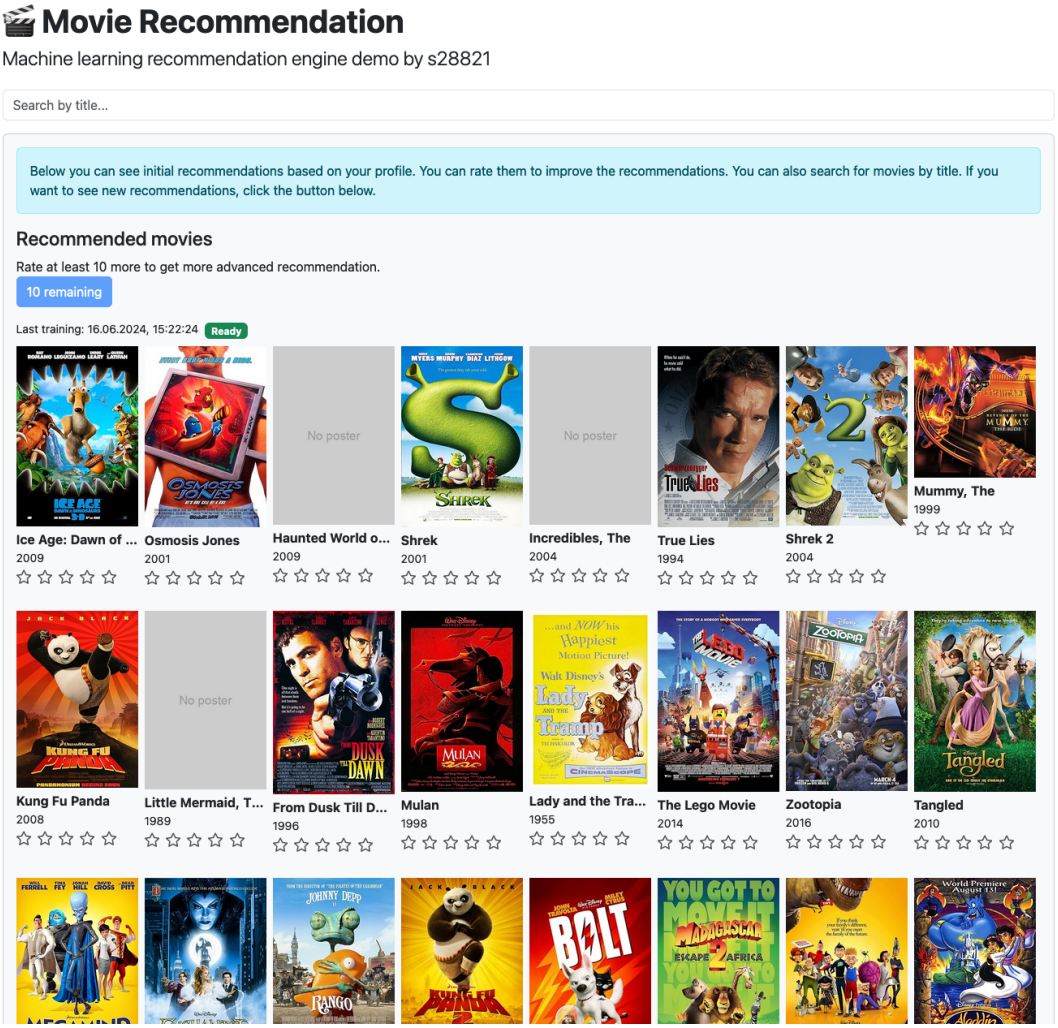


Rysunek 6.3: Zrzut ekranu prezentujący krok z wyborem okazji



Rysunek 6.4: Zrzut ekranu prezentujący krok z wyborem kategorii

Po wstępnym wyborze użytkownik widzi dobrane propozycje w oparciu o przypisane etykiety do filmów oraz wyborów dokonanych we wcześniejszych krokach. Na tym widoku użytkownik może ocenić filmy, które już obejrzał w skali od 1 do 5 gwiazdek gdzie 1 oznacza *Nie podobało mi się*, z kolei 5 oznacza *Bardzo mi się podobało*. Wstępne rekomendacje przedstawiono na rysunku 6.5.



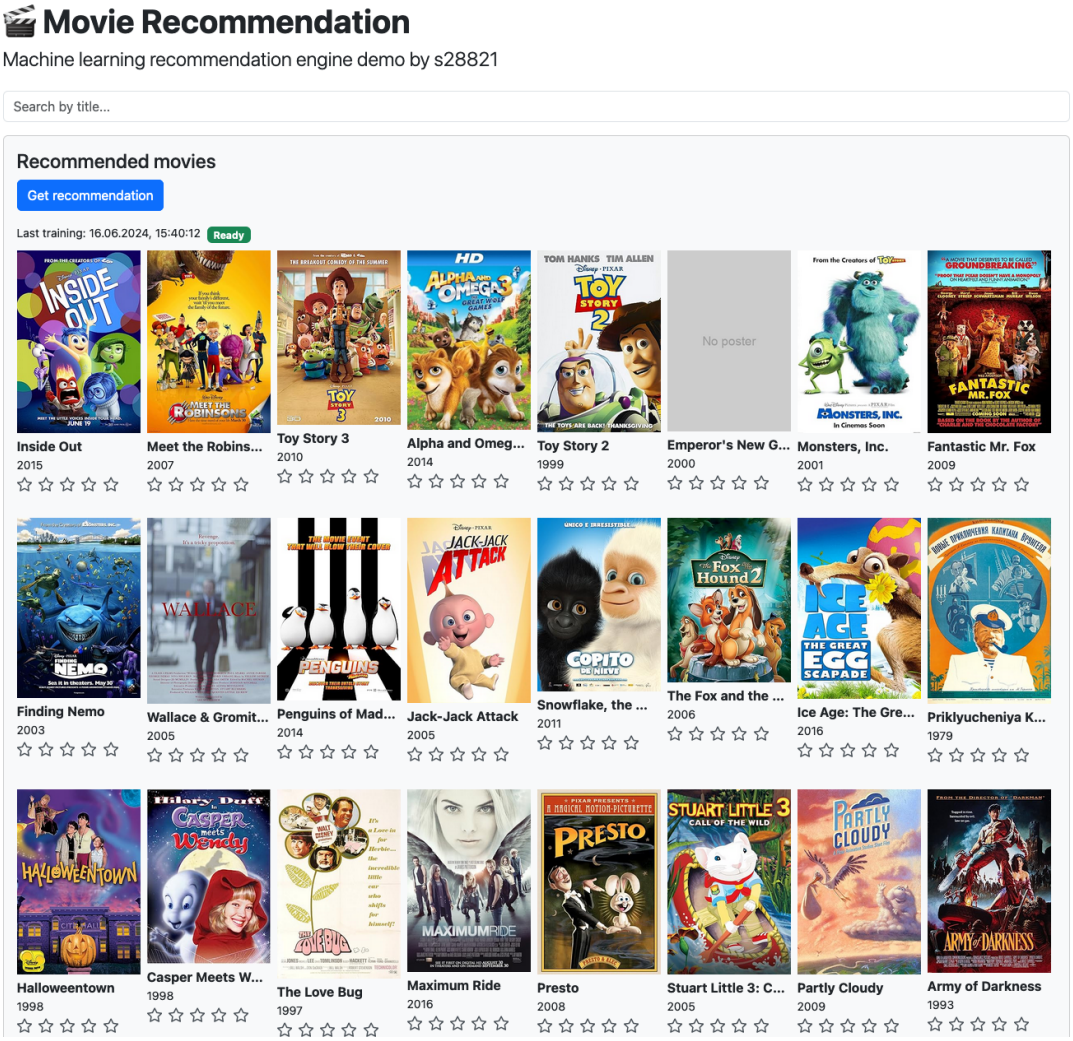
Rysunek 6.5: Zrzut ekranu ze wstępnymi rekomendacjami

W oparciu o uprzednio pozyskane oceny zwracane są dokładniejsze rekomendacje. Fragment 6.2 prezentuje wycinek logów, generowanych przez system rekomendacji w trakcie obsługi żądań od aplikacji klienckiej.

Kod 6.2: Logi z działającej usługi

```
127.0.0.1 - - [16/Jun/2024 15:40:09] "POST /api/ratings HTTP/1.1" 200 -
127.0.0.1 - - [16/Jun/2024 15:40:11] "POST /api/ratings HTTP/1.1" 200 -
127.0.0.1 - - [16/Jun/2024 15:40:11] "GET /api/ratings/status HTTP/1.1" 200 -
Reloading strategies
Reading config from /Users/localuser/recommendation/src/plugins/strategies.json
Loaded 2 strategies
Getting fresh data from database
Training models started
Training model from strategy <plugins.svd_strategy.SvdStrategy object at 0x10515f760>
127.0.0.1 - - [16/Jun/2024 15:40:12] "POST /api/ratings HTTP/1.1" 200 -
Training model from strategy <plugins.knn_strategy.KnnStrategy object at 0x10515fd90>
Computing the msd similarity matrix...
Done computing similarity matrix.
Models trained. Training took 0.6877281665802002 seconds
```

Gdy serwer wytrenuje model i użytkownik kliknie na przycisk *Get recommendation* zostaną zwrócone nowe rekomendacje z uwzględnieniem oddanych głosów jak pokazano na rysunku 6.6.



Rysunek 6.6: Zrzut ekranu z dokładnymi rekomendacjami

Aplikacja oferuje użytkownikom możliwość zawężenia poszukiwań poprzez wyszukiwarkę, zlokalizowaną w górnej części interfejsu aplikacji. Funkcjonalność ta stanowi uzupełnienie wbudowanego mechanizmu rekomendacji, dostarczając dodatkowe narzędzie do wyszukiwania wcześniej obejrzanych filmów by móc je także ocenić. Należy zauważyć, iż finalny rezultat jest związany z liczbą oddanych głosów. Im więcej głosów zostanie oddanych tym dokładniejsze będą finalne wyniki.

6.3 Analiza wykorzystania zasobów

Testy wydajnościowe przeprowadzono na platformie sprzętowej Macbook Pro, wyposażonej w procesor M2 Pro oraz 32 GB pamięci RAM. System operacyjny stanowił macOS w wersji 14.5, zoptymalizowany pod kątem architektury ARM64. Proces trenowania dwóch modeli, SVD oraz KNNBasic, dla zestawu danych MovieLens 100k [46], zajął 0,645 sekundy.

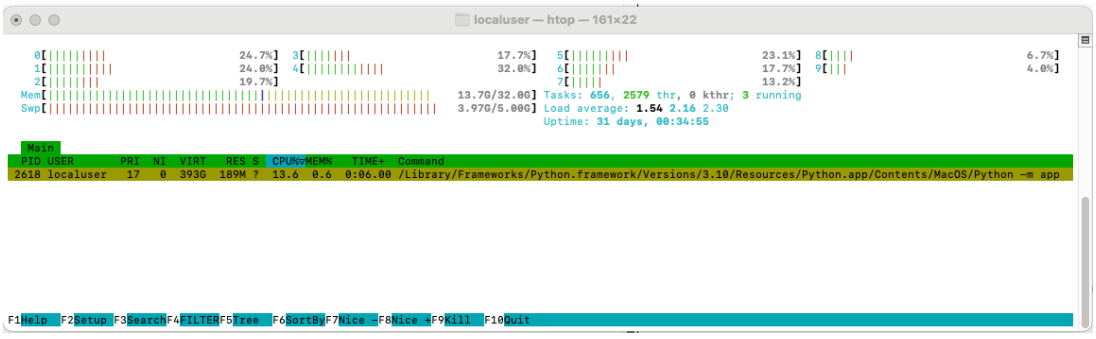
Monitorowanie zużycia zasobów systemowych przez badaną aplikację wykazało umiarkowane zapotrzebowanie na pamięć fizyczną, oscylujące w zakresie od 136,8 MB do 155,6 MB. Wynik pokazano w kodzie 6.3. Pomiary zostały zrealizowane z wykorzystaniem narzędzia `sample`, zintegrowanego z

systemem operacyjnym macOS. Narzędzie to pobierało próbki użycia pamięci przez proces Pythona z częstotliwością 1 ms, w czasie około 10 sekund. Zaobserwowano stabilność zużycia pamięci w analizowanym przedziale czasowym, co wskazuje na poprawność zarządzania tym zasobem przez testowaną aplikację.

Kod 6.3: Wynik monitorowania aplikacji z wykorzystaniem narzędzia `sample`

```
Analysis of sampling Python (pid 2554) every 1 millisecond
Process:       Python [25ww54]
Path:          /Library/Frameworks/Python.framework/Versions/3.10/Resources/Python.app/
    Contents/MacOS/Python
Load Address:  0x1006e8000
Identifier:    Python
Version:       ???
Code Type:     ARM64
Platform:      macOS
Parent Process: zsh [894]

Date/Time:     2024-06-16 17:18:07.364 +0200
Launch Time:   2024-06-16 17:17:57.437 +0200
OS Version:    macOS 14.5 (23F79)
Report Version: 7
Analysis Tool: /usr/bin/sample

Physical footprint:     136.8M
Physical footprint (peak): 155.6M
Idle exit:              untracked
```

W celu określenia stopnia wykorzystania procesora, posłużono się programem `htop`. Narzędzie to umożliwia monitorowanie obciążenia poszczególnych rdzeni procesora oraz określenie, które procesy mają największy wpływ na jego wydajność. Po zastosowaniu filtru, ograniczającego wyświetlane dane do procesów zawierających w nazwie frazę `python`, uzyskano rezultat zilustrowany na rysunku 6.7.



Rysunek 6.7: Zużycie zasobów według `htop`

Obciążenie procesora utrzymywało się na poziomie około 13,6% w momencie generowania rekomendacji oraz około 0,1% w spoczynku.

7. Podsumowanie

Systemy rekomendacji treści odgrywają istotną rolę we współczesnym świecie, gdzie ilość dostępnych informacji i produktów stale rośnie. Celem niniejszej pracy magisterskiej było zbadanie stanu sztuki, przegląd różnych algorytmów, metod i modeli generowania rekomendacji z uwzględnieniem ich możliwości, ograniczeń oraz zastosowań biznesowych. Celem było również stworzenie prototypu mikrousługi serwera rekomendacji treści.

W ramach pracy przedstawiono różne podejścia, algorytmy i modele służące pośrednio lub bezpośrednio do tworzenia systemów rekomendacji. Omówiono główne metody implementacji, w tym modelowanie oparte na treści, kolaboracyjne filtrowanie oraz modele hybrydowe. Zaprezentowano również przegląd dostępnych narzędzi i bibliotek języka programowania Python 3 - Scikit Learn [96], Surprise [101] i NLTK [19], które usprawniają proces tworzenia spersonalizowanych rozwiązań. Podkreślono interdyscyplinarny charakter problematyki, wskazując na konieczność współpracy specjalistów z różnych dziedzin, takich jak uczenie maszynowe, przetwarzanie danych, inżynieria oprogramowania oraz analiza biznesowa.

Utworzony w ramach pracy dyplomowej prototyp mikrousługi serwera rekomendacji treści stanowi rozwiązanie gotowe do wdrożenia za pomocą obrazu z platformy DockerHub [31]. Wykorzystane technologie, takie jak Python 3, Flask [79], PostgreSQL [102] i Docker, zapewniają skalowalność, wydajność i konteneryzację. Zastosowanie wzorców projektowych i architektonicznych, w tym MVC i wstrzykiwania zależności, przyczynia się do modularności i łatwości rozbudowy systemu. Mechanizm dynamicznego ładowania modułów z algorytmami uczenia maszynowego umożliwia dostosowywanie systemu do zmieniających się potrzeb bez konieczności przerywania jego pracy. Przeprowadzone testy jednostkowe oraz integracja z aplikacją kliencką potwierdzają poprawność działania mikrousługi oraz jej gotowość do obsługi rzeczywistego ruchu użytkowników. Udostępnienie obrazu aplikacji na platformie DockerHub ułatwia jej wdrażanie i skalowanie w różnych środowiskach.

Dzięki wybraniu danych MovieLens [46] jako podstawę do budowy i testowania systemu, zapewniono dostęp do obszernego i zróżnicowanego zbioru informacji o użytkownikach, filmach i ocenach. Dzięki reprezentatywności danych MovieLens, odzwierciedlających rzeczywiste interakcje i preferencje użytkowników, możliwe było przetestowanie różnych modeli rekomendacji. Proces ewaluacji pozwolił na ocenę skuteczności poszczególnych podejść oraz dostrojenie parametrów w celu uzyskania optymalnych wyników. Wykorzystanie tego zbioru danych zwiększyło wiarygodność uzyskanych rezultatów, podkreślając wartość przeprowadzonych badań dla rozwoju systemów rekomendacji.

Przedstawiony w tej pracy prototyp choć stworzony pierwotnie z myślą o danych MovieLens może z powodzeniem znaleźć swoje zastosowanie zarówno w sektorze e-commerce, gdzie może służyć do rekomendacji produktów popularnych wśród klientów danego typu lub sugerowania komplementarnych towarów. Również usługi bazujące na subskrypcji lub ruchu użytkownika mogą skorzystać z przygotowanego rozwiązania aby zatrzymać użytkownika na stronie lub w aplikacji dłużej. Ta niezależność od tematu wynika z przygotowania systemu od początku w myśl, która nie narzuca dziedziny biznesowej użycia.

W przygotowaniu rozwiązania przydatna była głównie literatura *Recommender Systems Handbook, Third Edition* autorstwa Francesco Ricci, Lior Rokach oraz Bracha Shapira [92], która poruszyła większość problemów związanych z projektowaniem, wdrożeniem oraz testowaniem systemów rekomendacji treści. Również *Practical Recommender Systems* [37], książka autorstwa Kima Falk była przydatna. Poza ogólnymi informacjami na temat budowy, wspominała również o biznesowych przypadkach, ograniczeniach i wymogach z nimi związanymi.

Inspiracją do podjęcia tematu systemów rekomendacji treści w niniejszej pracy magisterskiej był gwałtowny rozwój sztucznej inteligencji, obserwowany podczas codziennej pracy w firmie Brainly [3], dostarczającej rozwiązania edukacyjne zarówno dla klientów indywidualnych czyli uczniów i studentów, jak i klientów firmowych czyli wydawnictw naukowych. Chęć zgłębienia sposobu działania i metod

implementacji sztucznej inteligencji skłaniała do uczestnictwa w kursach i podejmowania zadań związanych z tą dziedziną. W konsekwencji również tematyka niniejszej pracy dyplomowej została osadzona w technologiach sztucznej inteligencji, a systemy rekomendacji treści, łączące aspekty uczenia maszynowego i rozwiązywania problemów biznesowych, okazały się optymalnym wyborem dla eksploracji tego zagadnienia. Praca magisterska pozwoliła na wzbogacenie wiedzy w dziedzinie uczenia maszynowego, programowania w języku Python 3 oraz poznanie metod i bibliotek służących do czyszczenia, wizualizacji i przetwarzania danych, takich jak Scikit Learn, Surprise i NLTK.

Zaprezentowany przegląd stanu sztuki, algorytmów, implementacji, rezultaty badań i stworzony prototyp systemu wraz z przykładową integracją, stanowią wprowadzenie do praktycznego zastosowania, dalszego rozwoju i adaptacji systemów rekomendacji w realnym środowisku biznesowym.

Spis rysunków

Spis tabel

Spis kodu

Prace cytowane

[1] Alpine Linux. https://alpinelinux.org/about/. [dostęp: 08.06.2024].

[2] Bootstrap Build fast, responsive sites with Bootstrap. https://getbootstrap.com/. [dostęp: 16.06.2024].

[3] Brainly - Learning, Your Way. https://brainly.com/. [dostęp: 17.06.2024].

[4] Docker Official Image of Python. https://hub.docker.com/_/python/. [dostęp: 08.06.2024].

[5] React - The library for web and native user interfaces. https://react.dev/. [dostęp: 16.06.2024].

[6] Stemming algorithms. https://snowballstem.org/. [dostęp: 15.04.2024].

[7] The Porter Stemming Algorithm. https://tartarus.org/martin/PorterStemmer/. [dostęp: 15.04.2024].

[8] TypeScript is JavaScript with syntax for types. https://www.typescriptlang.org/. [dostęp: 16.06.2024].

[9] unittest.mock — mock object library. https://docs.python.org/3/library/unittest.mock.html. [dostęp: 09.06.2024].

[10] Vite Next Generation Frontend Tooling. https://vitejs.dev/. [dostęp: 16.06.2024].

[11] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[12] H. Abdi and L. J. Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[13] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8):1295, 2020.

[14] Amazon Web Services, Inc. Amazon Web Services. https://aws.amazon.com/. [dostęp: 12.05.2024].

[15] Amazon.com, Inc. Net Perceptions Transforms World's Largest Bookstore into a Unique Personalized Shopping Experience. https://press.aboutamazon.com/1997/2/net-perceptions-transforms-worlds-largest-bookstore-into-a-unique-personalized-shopping-experience-grouplens-toolkit-gives-businesses-like-amazon-com-the-power-to-know-individual-customers-needs-and-increase-customer-satisfaction, 1997. [dostęp: 28.03.2023].

[16] Prafulla Bafna, Dhanya Pramod, and Anagha Vaidya. Document clustering: Tf-idf approach. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 61–66. IEEE, 2016.

[17] James Bennett and Stan Lanning. The Netflix Prize, 2007.

[18] Daniel Berrar et al. Cross-validation., 2019.

[19] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, 2006.

[20] Ekaba Bisong and Ekaba Bisong. Introduction to scikit-learn. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 215–229, 2019.

[21] David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. *The Journal of Machine Learning Research*, 3:601–608, 01 2001.

[22] Bridget Aymar. Fine Tuning the Social Web: John Riedl. `https://research.umn.edu/inquiry/post/fine-tuning-social-web-john-riedl`, 2013. [dostęp: 28.03.2023].

[23] Rasmus Bro and Age K Smilde. Principal component analysis. *Analytical methods*, 6(9):2812–2831, 2014.

[24] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12:331–370, 2002.

[25] Linguistic Data Consortium. Ontonotes release 5.0. `https://catalog.ldc.upenn.edu/LDC2013T19/`. [dostęp: 15.04.2024].

[26] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*. Google, Mountain View, CA, 2016.

[27] Daniele Varrazzo, The Psycopg Team. PostgreSQL driver for Python. `https://www.psycopg.org/`. [dostęp: 28.03.2023].

[28] John Deacon. Model-view-controller (mvc) architecture. *Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf*, 28:61, 2009.

[29] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc' aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc Le, and Andrew Ng. Large scale distributed deep networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[30] Dipanjan Sarkar. Implementing Deep Learning Methods and Feature Engineering for Text Data: The Continuous Bag of Words (CBOW). `https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-cbow.html`. [dostęp: 15.04.2024].

[31] Docker, Inc. Build and Ship any Application Anywhere. `https://hub.docker.com/`. [dostęp: 28.03.2023].

[32] Zhenhua Dong, Zhe Wang, Jun Xu, Ruiming Tang, and Jirong Wen. A Brief History of Recommender Systems, 2022.

[33] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.

[34] Erik Bernhardsson. Approximate Nearest Neighbors. `https://github.com/spotify/annoy`. [dostęp: 12.05.2024].

[35] Maria Eriksson, Rasmus Fleischer, Anna Johansson, Pelle Snickars, and Patrick Vonderau. Spotify teardown. *Inside the Black Box of Streaming Music*, 2019.

[36] Matthew Honnibal Explosion. spacy. `https://spacy.io/models/pl#pl_core_news_md/`. [dostęp: 15.04.2024].

[37] Kim Falk. *Practical Recommender Systems*. Manning, 2019.

[38] Asja Fischer and Christian Igel. An introduction to restricted boltzmann machines. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 17th Iberoamerican Congress, CIARP 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings 17*, pages 14–36. Springer, 2012.

[39] Jaques Grobler Gaël Varoquaux. The iris dataset. `https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html`. [dostęp: 26.05.2024].

[40] Aurélien Géron. *Uczenie maszynowe z użyciem Scikit-Learn, Keras i TensorFlow. Wydanie III*. Helion, 2023.

[41] GitHub, Inc. Let's build from here. `https://github.com/`. [dostęp: 28.03.2023].

[42] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, dec 1992.

[43] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[44] Cristos Goodrow. On youtube's recommendation system. `https://blog.youtube/inside-youtube/on-youtubes-recommendation-system/`. [dostęp: 11.05.2024].

[45] Konrad Gos and Wojciech Zabierowski. The comparison of microservice and monolithic architecture. In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 150–153. IEEE, 2020.

[46] GroupLens Research. Non-commercial, personalized movie recommendations. `https://movielens.org/`, 2023. [dostęp: 28.03.2023].

[47] S Hasanzadeh, Seyed Mostafa Fakhrahmad, and M Taheri. Based recommender systems: A proposed rating prediction scheme using word embedding representation of reviews. *The Computer Journal*, 65(2):345–354, 2022.

[48] Timothy O Hodson. Root mean square error (rmse) or mean absolute error (mae): When to use them or not. *Geoscientific Model Development Discussions*, 2022:1–10, 2022.

[49] Nicolas Hug. Surprise: A python library for recommender systems. *Journal of Open Source Software*, 5(52):2174, 2020.

[50] Chip Huyen. *Designing Machine Learning Systems*. O'Reilly Media, Inc., May 2022.

[51] Dietmar Jannach and Himan Abdollahpouri. A survey on multi-objective recommender systems. *Frontiers in Big Data*, 6, 2023.

[52] Casey Johnston. Netflix never used its $1 million algorithm due to engineering costs. *Ars Technica*, Apr 2012.

[53] Jean Kaddour, Aengus Lynch, Qi Liu, Matt J Kusner, and Ricardo Silva. Causal machine learning: A survey and open problems. *arXiv preprint arXiv:2206.15475*, 2022.

[54] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[55] Kenrick Davis. The App That Launched a Thousand Memes. `https://www.sixthtone.com/news/1001728`. [dostęp: 12.05.2024].

[56] Osman Khalid, Samee U. Khan, and Albert Y. Zomaya, editors. *Big Data Recommender Systems - Volume 1: Algorithms, Architectures, Big Data, Security and Trust*. Big Data Recommender Systems. IET, Institution of Engineering and Technology, 2019.

[57] Kong Inc. The easy way to design, debug, and test APIs. `https://insomnia.rest/`. [dostęp: 28.03.2023].

[58] Joseph A Konstan, John Riedl, Al Borchers, and Jonathan L Herlocker. Recommender systems: A grouplens perspective. In *Recommender Systems: Papers from the 1998 Workshop (AAAI Technical Report WS-98-08)*, pages 60–64. AAAI Press Menlo Park, 1998.

[59] Yehuda Koren. The bellkor solution to the netflix grand prize. Aug 2009. Introduction to BellKor's Pragmatic Chaos' final solution for the Netflix Grand Prize, including improvements and new predictors.

[60] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

[61] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2nd international conference on Ubiquitous information management and communication*, pages 208–211, 2008.

[62] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Proceedings fifth ieee international enterprise distributed object computing conference*, pages 118–127. IEEE, 2001.

[63] Juha Leino and Kari-Jouko Räihä. Case amazon: ratings and reviews as part of recommendations. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 137–140, 2007.

[64] David Leonhardt. You want innovation? offer a prize. *The New York Times*, 30, 2007.

[65] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyun Ni, and Ning Wang. The design and implementation of a real time visual search system on jd e-commerce platform, 2019.

[66] Joseph Lilleberg, Yun Zhu, and Yanqing Zhang. Support vector machines and word2vec for text classification with semantic features. In *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*, pages 136–140. IEEE, 2015.

[67] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003.

[68] Zhuoran Liu, Leqi Zou, Xuan Zou, Caihua Wang, Biao Zhang, Da Tang, Bolin Zhu, Yijie Zhu, Peng Wu, Ke Wang, and Youlong Cheng. Monolith: Real time recommendation system with collisionless embedding table. *Bytedance Inc.*, 2022. Bolin Zhu: Fudan University.

[69] Alexander Ljungqvist and William J Wilhelm Jr. Ipo pricing in the dot-com bubble. *The journal of Finance*, 58(2):723–752, 2003.

[70] Julie Beth Lovins. Development of a stemming algorithm. *Mech. Transl. Comput. Linguistics*, 11(1-2):22–31, 1968.

[71] Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836, 2018.

[72] Matthew Honnibal Explosion. spaCy. `https://github.com/explosion/spaCy`. [dostęp: 15.04.2024].

[73] Rachana Mehta and Keyur Rana. A review on matrix factorization techniques in recommender systems. In *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, pages 269–274. IEEE, 2017.

[74] Michał Żmuda and Joanna Misztal-Radecka and Piotr Turek. Why Choose User Segmentation for Your Recommender System in 2020? `https://tech.ringieraxelspringer.com/blog/big-data-and-machine-learning/why-choose-user-segmentation-for-your-recommender-system-in-2020/b7q4488/`, 2020. [dostęp: 30.03.2023].

[75] Neo4j, Inc. Neo4j Graph Data Platform. `https://neo4j.com/`. [dostęp: 28.03.2023].

[76] Netflix, Inc. Netflix Prize Leaderboard. `https://web.archive.org/web/20131213101429/http://www.netflixprize.com//leaderboard?showtest=t&limit=20/`, 2009. [dostęp: 28.03.2023].

[77] New York Times. At&t's history of invention and breakups. *The New York Times*, Feb 2016.

[78] Hoang Nguyen, Rachel Richards, C.-C. Chan, and Kathy J. Liszka. Redtweet: Recommendation engine for reddit. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1381–1388. IEEE/ACM, 2015.

[79] Pallets. The Python micro framework for building web applications. `https://flask.palletsprojects.com/en/3.0.x/`. [dostęp: 28.03.2023].

[80] Instytut Podstaw Informatyki PAN. Narodowy korpus języka polskiego. `https://nkjp.pl/`. [dostęp: 15.04.2024].

[81] Tulasi K Paradarami, Nathaniel D Bastian, and Jennifer L Wightman. A hybrid recommender system using artificial neural networks. *Expert Systems with Applications*, 83:300–313, 2017.

[82] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[83] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[84] PostgreSQL Global Development Group. libpq. `https://www.postgresql.org/docs/current/libpq.html`. [dostęp: 01.06.2024].

[85] Dhananjay Prasanna. *Dependency injection: design patterns using spring and guice*. Simon and Schuster, 2009.

[86] Pydantic People. Pydantic is the most widely used data validation library for Python. `https://flask.palletsprojects.com/en/3.0.x/`. [dostęp: 28.03.2023].

[87] Python Software Foundation. Python. https://www.python.org/. [dostęp: 28.03.2023].

[88] Dong Qiu, Haihuan Jiang, and Shuqiao Chen. Fuzzy information retrieval based on continuous bag-of-words model. *Symmetry*, 12(2):225, 2020.

[89] RecoAI. Netflix recommendation system: How it works. https://recoai.net/netflix-recommendation-system-how-it-works/, 2022. [dostęp: 28.03.2023].

[90] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[91] Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40(3):56–58, mar 1997.

[92] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook, Third Edition*. Springer, 2022.

[93] Ringier Axel Springer Polska sp. z o.o. We empower free decisions. https://ringieraxelspringer.pl/. [dostęp: 30.03.2023].

[94] Seok-Beom Roh and Tae-Chon Ahn. Design of lazy classifier based on fuzzy k-nearest neighbors and reconstruction error. *Journal of the Korean Institute of Intelligent Systems*, 20(1):101–108, 2010.

[95] Michael Schrage. *Recommendation Engines*. MIT Press Essential Knowledge series. The MIT Press, 2020.

[96] scikit-learn. Machine Learning in Python. https://scikit-learn.org/stable/. [dostęp: 28.03.2023].

[97] Amit Sharma, Jake M. Hofman, and Duncan J. Watts. Estimating the causal impact of recommendation systems from observational data. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15. ACM, June 2015.

[98] Brent Smith and Greg Linden. Two decades of recommender systems at amazon.com. *IEEE Internet Computing*, 21(3):12–18, 2017.

[99] Steve Lohr. Netflix Cancels Contest After Concerns Are Raised About Privacy. https://www.nytimes.com/2010/03/13/technology/13netflix.html, 2010. [dostęp: 28.03.2023].

[100] Gilbert W Stewart. On the early history of the singular value decomposition. *SIAM review*, 35(4):551–566, 1993.

[101] Surprise. A Python scikit for recommender systems. https://surpriselib.com/. [dostęp: 28.03.2023].

[102] The PostgreSQL Global Development Group. PostgreSQL: The World's Most Advanced Open Source Relational Database. https://www.postgresql.org/. [dostęp: 28.03.2023].

[103] Jørgen Thingstad. The impact of spotify's ai-driven music recommender on user listener habits, 2023.

[104] Princeton University. Wordnet. https://wordnet.princeton.edu/. [dostęp: 15.04.2024].

[105] Eliot Van Buskirk. Bellkor's pragmatic chaos wins $1 million netflix prize by mere minutes. *Wired*, Sep 2009.

[106] Michelle Williams. 'this internet thing is not a fad': Reddit co-founder alexis ohanian to discuss online entrepreneurship at umass amherst, 2014. [dostęp: 12.05.2024].

[107] Street Nick Wolberg William, Mangasarian Olvi and Street W. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C5DW2B.

[108] Andrew Zhai. Representation learning for recommender systems, Aug 2021. [dostęp: 15.04.2024].

[109] Min Zhang and Yiqun Liu. A commentary of tiktok recommendation algorithms in mit technology review 2021. *Fundamental Research*, 1(6):846–847, 2021.

Dodatki

Na płycie CD dołączonej wraz z pracą przekazano:

- Kod źródłowy prototypu systemu rekomendacji.
- Dokument źródłowy w formacie LaTeX z pracą magisterską.